COMPUTATIONAL MODELING AND SIMULATION OF COMPOSITE MATERIALS BASED ON TOMOGRAPHIC IMAGES

BY

ELENA CARABA

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2014

Urbana, Illinois

Doctoral Committee:

Professor Michael Heath, Chair
Professor Philippe Geubelle
Professor John Hart
Associate Professor Luke Olson
Associate Professor Xiangmin Jiao, Stony Brook University

Abstract

The goal of this work is to enable automated thermal and mechanical finite element analysis of heterogeneous composite materials based on tomographic images of material specimens. Of particular interest are microvascular materials containing embedded micro-channnels through which a coolant or self-healing agent can be circulated. Such materials have important applications to aircraft, electronics, and many other situations where active management of the thermal environment is desirable.

Conventional computational analysis of such materials is complicated by their complex geometry and heterogeneous material properties. In particular, conventional finite element analysis requires highly refined, unstructured meshes that conform to the complex geometry, resulting in high computational cost. We take a different approach based on the Interface Generalized Finite Element Method (IGFEM), in which a structured and relatively coarse finite element mesh is used, with the complex geometry incorporated by means of interface surfaces that intersect elements of the mesh. The discretized solution space is then augmented by enrichment functions associated with points of intersection with material interfaces, thereby enabling the accurate capture of discontinuities in the solution gradient along material interfaces.

An important feature of our approach is the use of 2D or 3D tomographic images of actual material specimens to determine the complex geometry. Based on such images, we generate a pixel- or voxel-aligned rectangular mesh that is selectively refined to attain the necessary accuracy near material interfaces, as well as satisfying constraints imposed by the finite element methodology. Although quadrilateral or hexahedral elements are obviously most natural in our context, IGFEM has primarily been used only with triangular elements in 2D and tetrahedral elements in 3D because of the much simpler implementation of the

intersections of interfaces with mesh elements. The greater complexity of IGFEM using hexahedral meshes becomes more manageable, however, in the context of pixel- or voxel-aligned meshes and pixel- or voxel-defined interfaces.

We first consider the 2D case, for which we develop an adaptive mesh generation algorithm as well as an implementation of IGFEM for performing the subsequent analysis. The effectiveness of both is demonstrated by solving a thermal test problem for various geometries inferred from 2D images of heterogenous materials, some artificially generated and others 2D slices of 3D tomographic images of real heterogeneous materials. Results on convergence and complexity are provided, along with comparisons with the conventional finite element method. We then extend the adaptive mesh generation algorithm to the 3D case, addressing the additional constraints and complications that arise in this context. Finally, we show results for a series of complex geometries given by 3D tomographic images, both artificially and experimentally generated.

To my mom.

Acknowledgments

I am grateful to the following mentors, friends and family for their guidance, support and love.

- My advisor, Prof. Michael Heath for guiding me through this project and constantly offering me his feedback. His advice and critique have helped me become a better researcher. Thank you for being the best advisor anyone could ask for.
- Prof. Philippe Geubelle, my co-advisor, for opening the path to a collaborative interdisciplinary research, for his guidance, and for his insight and dedication to interface-based generalized finite element method.
- My committee members, Prof. Luke Olson, Prof. John Hart and Prof. Xiangmin Jiao for providing valuable feedback for my research.
- Prof. Paul Saylor for introducing me to Numerical Analysis, for being a wonderful mentor and friend. Cynthia Saylor for convincing me to join UIUC and for her caring love and support.
- Prof. Gabrielle Allen and Prof. Edward Seidel for their mentorship and friendship throughout the years, for being the first to expose me to research.
- Dr. Wes Bethel, Dr. Cecilia Aragon, Dr. Raquel Romano, Dr. Mark Hereld, Dr. Michael Papka, and Dr. Richard Lehoucq, for honing my research skills during summer internships.
- Dr. Masoud Safdari and Dr. Soheil Soghrati for their valuable input and numerous discussions on the finite element method.

- Jason Patrick, Joe Gonzalez and Sang Kim for helping me obtain 3D real data sets of their composite materials.

- Huong Luu, Steve Dalton, Jehanzeb Hameed, Hormozd Ghavari, Nana Arizumi, Mark Gates, Peng Jiang, Jayanta Mukherjee, Kaushik Kalyanaraman, Matthew Michelotti and Amanda Bienz for sharing the graduate experience with me.

- My parents and my sisters for their unconditional love and belief in me, and in particular my mother for being my biggest supporter.

- My maternal grandmother for her sacrifice and hard work for better access to good education. You are my inspiration.

- Zack for being a wonderful and supportive husband right from the beginning, and Olivia for being my motivation.

TABLE OF CONTENTS

CHAPTER 1 Introduction 1
1.1 Background 1
1.2 Motivation 3
1.3 Fabrication Procedures for Composites 5
1.4 Experimental Setup 7
1.5 Noise 8
1.6 Thesis Objectives 10
1.7 Finite Element Software Framework 10
1.8 Thesis Organization 12

CHAPTER 2 2D Meshing Based on Image Data 13
2.1 Meshes 14
2.2 Available Meshing Software for Tomographic Data 16
2.3 Quadtree Algorithm 17
2.4 Indexing and Computing Neighbors 21
2.5 Test Images 26
2.6 IGFEM-Imposed Constraints 29
2.6.1 *hq*-refinement 30
2.6.2 Interface Constraint 33
2.6.3 1-irregular Rule 34
2.6.4 3-neighbor Rule 36
2.6.5 Steep-gradient Constraint 39
2.6.6 Final Constraints 42
2.7 Localization and Approximation of Material Interfaces 49
2.7.1 Interface Continuity 50
2.7.2 Polynomial Interpolation 50
2.7.3 Results 53
2.7.4 NURBS or Non-Uniform Rational B-Splines Interpolation 58

CHAPTER 3 Interface-based Generalized Finite Element Analysis with Quadrilateral Meshes 67
3.1 Problem Statement 68
3.2 Partition of Unity and Generalized/Extended Finite Element Methods . . . 70
3.3 IGFEM for Quadrilateral Meshes 72

3.3.1 Hanging Nodes . . . 74
3.3.2 Isoparametric Elements . . . 76
3.3.3 Integration . . . 85
3.3.4 Domains of Integration . . . 87
3.4 Heat-Conduction Test Problem . . . 93

CHAPTER 4 3D Meshing Based on Image Data . . . 102
4.1 Octree Algorithm . . . 103
4.2 Indexing and Computing Neighbors . . . 106
4.3 Test Images . . . 107
4.4 IGFEM Imposed Constraints . . . 110
4.4.1 Mesh Size Constraint . . . 110
4.4.2 Interface Constraint . . . 111
4.4.3 1-irregular Rule . . . 111
4.4.4 k-neighbor Rule . . . 112
4.4.5 Steep-gradient Constraint . . . 112
4.4.6 Final Constraints . . . 113
4.5 Localization and Approximation of Material Interfaces . . . 117
4.5.1 Planar Surface Approximation . . . 117
4.5.2 Non-Rational Uniform B-Splines in 3D . . . 118
4.5.3 Comparison of Planar and NURBS Approximations . . . 122

CHAPTER 5 Summary and Future Directions . . . 126
5.1 Summary . . . 126
5.2 Future Work . . . 129

REFERENCES . . . 131

CHAPTER 1

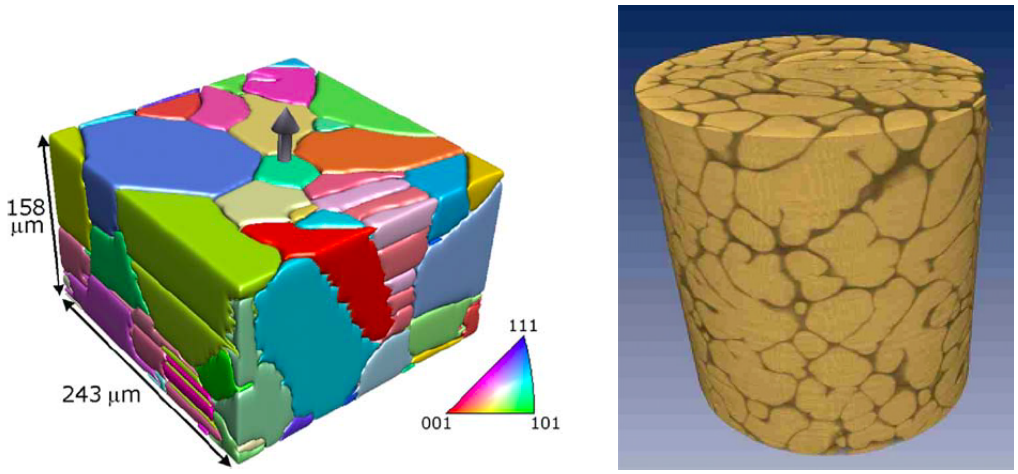
Introduction

1.1 Background

Many materials that play important roles in materials science and technology are heterogeneous, i.e., they consist of dissimilar "phases" that are distinguishable only at small length scales. Some well-known examples of such materials are composites, concrete, polycrystalline materials, porous and cellular materials, and bone (Figure 1.1).The main focus of our research is composites that mimic biophysical processes such as self-cooling or self-healing, modeled after circulatory systems found in many living organisms. Similar to vascular networks, microchannels ranging from 10 μm to 1 mm in diameter are embedded in a polymeric matrix.

A self-healing material has a healing agent encapsulated in its channels. Like a cut on human skin that triggers blood from the capillary network to flow to the wound site and rapidly form a clot to start the healing process, a crack on the surface of a material activates the liquid healing agent flowing through the microvascular network to be absorbed at the site of the damage [55].

A self-cooling material has a coolant flowing through its microchannels. In human skin, blood vessels thermally regulate temperature. When the outside temperature is increased or decreased, the vascular response of the skin is able to adjusts its temperature to preserve or dissipate heat. Similarly, the circulation of a coolant in microchannels can reduce the temperature of the surrouding material by direct extraction of heat from the thermally loaded material, and by redistributing the heat between warmer and cooler regions of the domain [51]. Some of the main engineering applications of autonomic healing and active cooling materials are hypersonic vehicles, micro-electronics and batteries.



(a) (b)

Figure 1.1: (a) Austenite phase in AL-6XN [35]. (b) Zirconia in bright, amorphous phase in dark [37].

Of high interest to our research is a self-cooling hybrid material used in the construction of hypersonic and reentry vehicles. Due to friction, shock waves, and radiation, the outer surface of airplanes during high-speed flights can cause temperatures to rise above 1000°C. No polymer can sustain that level of heating. A proposed alternative is a hybrid material with a ceramic plate on the hot side and composites on the cooler side. The temperatures can still exceed the allowable bounds, but incorporating active cooling can regulate them to a desired range. Traditional materials such as epoxy can be used by embedding within them a network of microvascular channels through which a coolant flows. In applications where there is no means for human intervention, such as manned or unmanned space vehicles, having autonomic functionality such as self-healing is very important. In case of damage due to fatigue or impact from debris, autonomic repair of the composite by circulating liquid healing agents can be achieved. Self-healing improves not only longevity of aerospace structures, but also their safety.

The design of a microvascular network is application specfic, and an optimal configuration of the microchannels requires an accurate and efficient thermo-mechanical analysis. Networks designed for flow efficiency may look very different from those designed for structural performance. The assumptions made in the representation of a microstructure can be a significant impediment to physical modeling to predict the response of the material. Thus,

the most accurate method to represent a 3D microstructure on a computer is to explicitly translate its volume collected through an experiment into a computational method. Some of the most common approaches in transforming physical materials to pixel/voxel data are through serial sectioning experiments that are usually coupled with electron backscattered diffraction. Serial sectioning, unfortunately, is a process that is destructive, time-consuming, and erratic. A good alternative is to use X-ray based techniques instead. These techniques are non-destructive, the samples remaining intact after the analysis.

Looking beyond our initial target materials, we demonstrate the broader applicability of our methods by considering other types of composite materials, such as fiber reinforced composites and battery electrodes.

1.2 Motivation

The behaviour of heterogeneous materials is determined by the underlying material properties of each phase, the underlying geometry, and the underlying topology corresponding to the phase arrangement. The availability and quality of this information affects the overall accuracy of the model.

Modeling the thermal or mechanical responses of microvascular materials is a complicated process defined by two major sources of complexity: one associated with presence of discontinuous gradient fields for the temperature and/or displacement fields, and another associated with the complex geometry of the microstructure.

Heterogeneous materials such as polycrystals have been observed to have C^0 continuity along grain boundaries, while composite materials have a discontinuous gradient along the boundaries of the inclusions. The discontinuity in the gradient also arises when applying thermal or structural loads over very narrow regions, as is the case with an embedded microvascular network in a polymer. The geometric complexity associated with this class of materials is especially prevalent when creating realistic models of the microstructure directly from tomographic data. Such data are obtained by performing a series of tomographic scans of a material sample that are then converted into a 3D array of voxels (analogous to an array of pixels in 2D), each with a numerical value representing a small portion of the material.

The classical approach in the numerical treatment of heterogeneous materials is the finite element method (FEM), due to its ability to treat both complex geometries and physical phenomena. FEM achieves acceptable accuracy on problems with discontinuous gradients by using meshes that conform to the internal geometry of the microstructure. The approximation satisfies the C^0 continuity in the jumps of the gradient along the boundary; and can accurately represent the gradient discontinuity between adjacent elements with distinct material properties. However, a conforming mesh is difficult to generate and is typically very large, yielding a complex and expensive solution process. The construction of such meshes becomes even more difficult for higher-dimensional problems, where one must account not only for interfaces but also for other features of the model, such as grain boundaries or inclusions. Thus, the conventional approach becomes too computationally expensive, or in some cases, infeasible. Another issue with conforming meshes is that the geometric representation of the problem may also require elements that have an unacceptable aspect ratio. A method that provides independence between the geometry of the problem and the finite element mesh is desirable for reducing complexity and time-to-solution. Such a method that can alleviate some of the geometric complexities, is the Generalized Finite Element Method (GFEM) [14], also referred to as the Extended Finite Element Method or XFEM [40]. GFEM has been introduced over the past decade or so to allow for numerical treatment of complex microsctructures using finite element discretizations that do not conform to the material interfaces. These methods capture the presence of gradient discontinuities along material interfaces by enriching the finite element approximation in the elements intersected by such interfaces.

Recent work [49, 50] extends the GFEM methodology by applying the basis functions at nodes defined by the intersection of the interface with the non-conforming finite element mesh. The result is a new method, called the Interface-based Generalized Finite Element Method (IGFEM), which has shown great promise in the analysis and computational design of microvascular materials [51] and the multiscale failure analysis of heterogeneous adhesives [8]. With IGFEM we put complexity into the method, rather than the mesh, by creating relatively coarse, structured, nonconforming meshes. The material interfaces are incorporated by enriching the finite element solution with additional basis functions at intersection points

between mesh elements and material interfaces. IGFEM and enrichment are discussed in detail in Chapter 3. The application of IGFEM has so far been limited to the treatment of heterogeneities with simple shapes, described explicitly through relatively simple mathematical expressions (such as sinusoidal functions for the case of embedded microchannels) and to meshes made of 2D (triangular) and 3D (tetrahedral) elements. To achieve higher level of accuracy in the modeling of more realistic material systems, in this thesis we adapt and extend IGFEM to model composite materials whose geometry is extracted directly from tomographic images.

The meshes used by finite element methods must meet specific requirements, such as various constraints imposed by the Galerkin formulation and discretization. IGFEM imposes additional constraints requiring development of new meshing techniques. In this thesis we develop a two-dimensional meshing algorithm that we then tested in conjunction with a two-dimensional implementation of the IGFEM analysis. We extended the current IGFEM formulation to quadrilateral meshes, as such meshes are natural to pixel-aligned datasets. We also develop a corresponding three-dimensional meshing algorithm that generates hexahedral meshes suitable for a three-dimensional implementation of IGFEM under development by another collaborating researcher.

1.3 Fabrication Procedures for Composites

Figure 1.2 shows the fabrication procedure for a microvascular fiber-reinforced composite through a process called vaporization of sacrificial components or VaSC [25]. The procedure starts with the mechanized weaving of straight warp and weft yarns (blue) with interwoven Z-fiber tows (green) shown in Figure 1.2a. Sacrificial fibers (red) are woven into the preform (Figure 1.2b) to form an orthogonal 3D structure. The position, length, diameter, and curvature of the fibers can be varied to meet certain design criteria. The space between fibers is infiltrated with a low-viscosity thermosetting resin (Figure 1.2c), such as epoxy, and cured at high temperatures, resulting in a 3D woven composite (Figure 1.2d). After being cured, the sample is trimmed to expose the ends of the sacrificial fiber that is then removed by heating the sample above 200°C for several hours. The high temperature vaporizes the

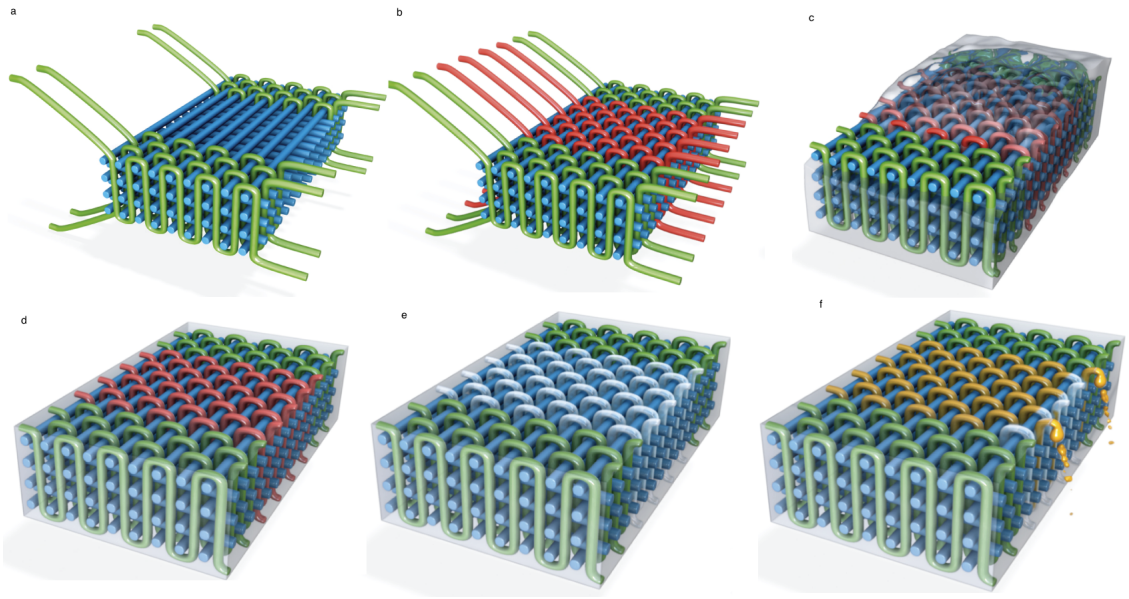
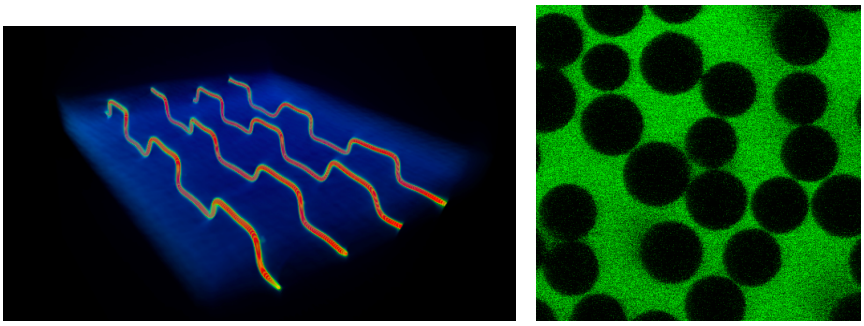


Figure 1.2: Fabrication procedure for a microvascular fiber-reinforced composite [25].

polylactide that impregnates the fiber, yielding a hollow channel that is a high-fidelity inverse replica of the original fiber's diameter and trajectory (Figure 1.2e). The network of empty channels is then filled with a fluid (e.g., coolant, organic solvents, liquid metals) having the desired properties (Figure 1.2f) to create a 3D microvascular composite that is both strong and multifunctional. The strength and form are provided by the solid phase of the new material, while the capability to adapt to multiple functionalities is provided by the liquid phase. The sacrificial fibers used in VasC must be strong enough to survive the mechanical weaving, and must remain solid during the matrix curing (up to 180°C), but disintegrate easily via depolymerization to monomer vapor at higher temperatures.

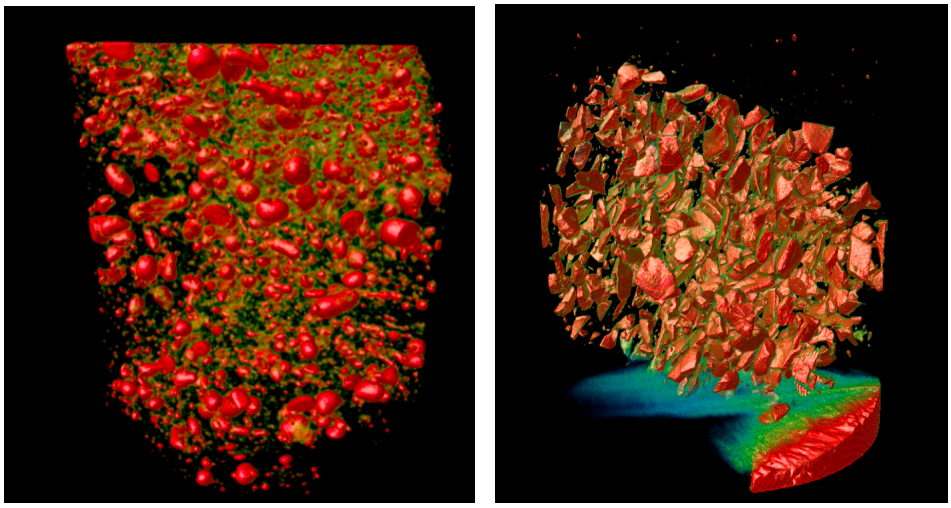
Figure 1.3a shows a 3D microvascular network of channels. Figure 1.3b shows a Glass-fiber reinforced composite.

Another material we consider is battery electrodes, made of a polymer matrix with either tin or silicon inclusions. Sn or Si powder was mixed with either a solvent or distilled water to form a viscous slurry, which was further homogenized with a planetary mixer and then heated in vacuum to remove excessive water. Figure 1.4 shows the Sn and Si inclusions in



(a) (b)

Figure 1.3: (a) 3D Micro-CT image of microvascular material [44]. (b) 2D slice of glass-fiber reinforced composite material [54].



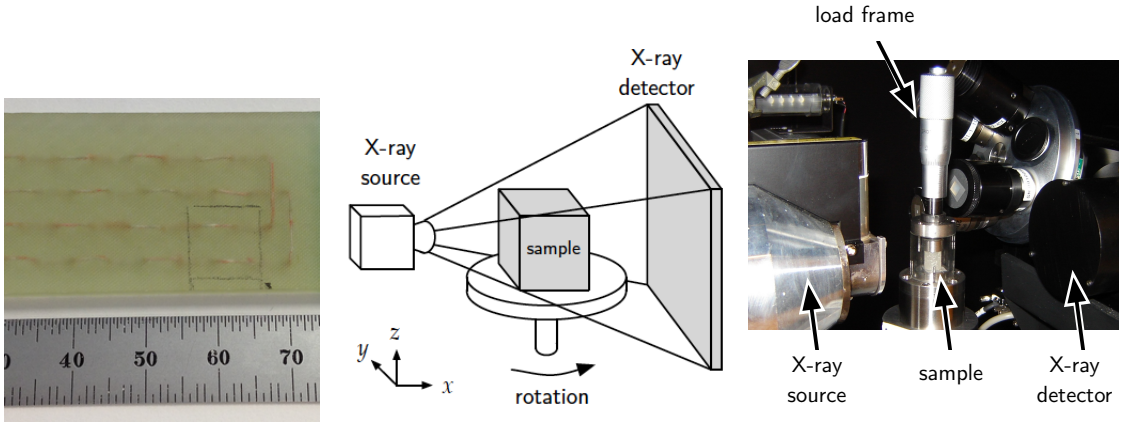
(a) Tin (b) Silicon

Figure 1.4: Battery electrodes in polymer matrix [29].

red, with the polymer matrix being obscured.

1.4 Experimental Setup

High energy X-ray tomography can reveal microstructural features of a material, such as dissimilar phases, cracks, pores, etc. The state of the art in working with real composite materials is to perform a series of tomographic scans of the sample material under study and then to convert the scans into a 3D voxel representation. This is achieved by recording multiple radiographs of the sample material at multiple angular positions, and then using



(a) (b) (c)

Figure 1.5: (a) Photograph of material.(b) Diagram of a CT scanner. (c) Xradia CT scanner interior with sample mounted inside load frame [Credit for (b), (c): Mark Gates]

these to reconstruct a 3D map of the X-ray attenuation coefficient within the material. The sample is affixed to a stage (Figure 1.5b), which the scanner rotates 360° about the vertical axis in fixed angular steps. As X-ray beams project, they create elements corresponding to line integrals of attenuation coefficients, resulting in a volume superimposed on a 2D plane. Multiple such radiographs created from different angular positions eventually result in a 3D image.

1.5 Noise

In a idealized world, computed tomography images would be a perfect reflection of reality. Unfortunately, low photon counts, low radiation dose, finite scanner resolution, motion and scatter usually corrupt the data to some degree and introduce artifacts.

In working with our materials and their tomographic representations, we have identified several sources of noise. One major source comes from the micro-CT machine itself, as the reconstructed images are often subjected to blur and noise due to light absorption/scattering, positioning of the samples, and image reconstruction algorithms. Buades et. al. [18] blame both the finite nature of images that results in blurring, and the number of photons and heating that result in noise. Unfortunately, taking repetitive scans of the same material does

not lead to the same data set. This is caused mainly by variability in pixel reading of the machine. The best approach is to average several scans and use this combined data set.

Scanner noise is not the only source of error in the pixel readings. An imperfect process of manufacturing of the material can lead to additional problems. If the resin infusing the fibers does not thoroughly cover the sacrificial fiber and leaves empty pockets around it, then after depolymerization, the hollow channel may no longer represent the perfect shape of the original fiber. These imperfections appear as smudges around the channel that can be reduced by pre-smoothing the image data. Similar imperfections were also observed when working with the battery and fibers datasets.

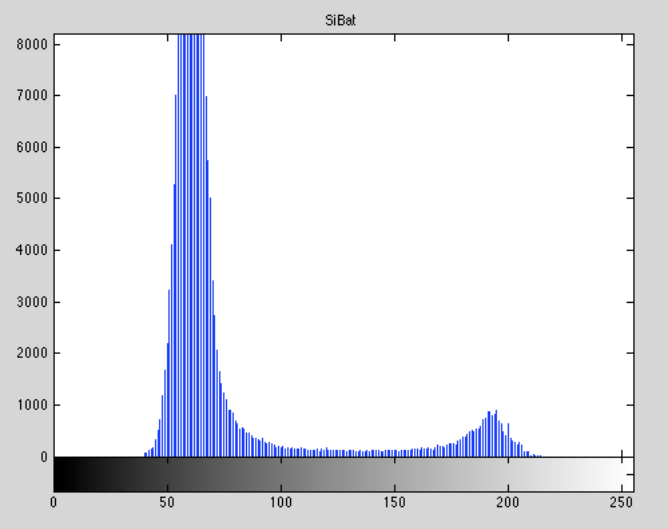


Figure 1.6: Pixel color distribution for slice of Silicon battery dataset.

All of these sources can be dealt with by applying denoising algorithms at various levels of representation of the micro-CT data. We categorize each type of material by assigning to it a range of pixel/voxel colors. We create a histogram showing the distribution of grayscale pixels/voxels contained in the image and observe the peaks present to determine boundaries for bins that contain an appropriate range of colors. Figure 1.6 shows a typical histogram. For this example, we could choose one bin with values between 0 and 150 to represent one type of material, and a second bin with values between 150 and 255 to represent the other material. Optionally, we may also pre-smooth the image data, then transform it into a binary image (i.e., having only two distinct pixel/voxel values) to delineate clearly between

the dissimilar materials in the image. Depending on the type of material, a final sweep through the dataset may optionally be made to remove any inclusions that are smaller than a given tolerance, assuming these would be deemed insignificant in subsequent analysis.

1.6 Thesis Objectives

The main objective of this thesis is to develop a methodology for automated meshing in 2D and 3D of heterogeneous materials based on their tomographic images. In particular, we are interested in creating realistic models of composite materials that have a discontinuous gradient along the boundaries of the inclusions (C^0 continuity). The resulting meshes will be problem geometry dependent rather than solution driven; they will be a subset of the pixel/voxel data set, relatively coarse, and adaptive, nonconforming to the material interfaces, and structured.

The goals can be summarized as follows.

- Develop and implement 2D and 3D algorithms to create structured, adaptive, and nonconforming quad/hex meshes.
- Determine interface presence, location, and representation based on pixel or voxel data.
- Incorporate mesh adaptivity driven by
  - how well interface is approximated using different numerical methods such as Newton polynomials, B-Spline interpolation, or least squares fit,
  - interface based generalized finite element imposed constraints.
- Test IGFEM on quadrilateral meshes by solving 2D thermal test problem for various composite material geometries.

1.7 Finite Element Software Framework

To test both our generated meshes and adaptation of IGFEM, we built a finite element software framework composed of three major steps: a *preprocessing module* that sets up the

problem, the solver, and the domain; a *processing module* that solves the partial differential equation; and a *postprocessing module* that assesses the solution quality and shows the final results.

Module 1: *Preprocessing*

- Create tomographic representation of material of interest.
- Set parameters for interface approximation such as the interpolation method and its degree, mesh element maximum and minimum size, and any required thresholds.
- Obtain geometric representation of material internal structure.
- Discretize domain into a structured, non-conforming, adaptive mesh.
- Set up thermal test problem: load vector, boundary conditions, conductivity coefficients, degree of basis functions.

Module 2: *Processing*

- Generate local stiffness matrices and load vectors.
- Assemble global stiffness matrix and load vector.
- Enforce boundary conditions.
- Solve assembled algebraic system for finite element solution.

Module 3: *Postprocessing*

- Assess quality of solution. Compute error norms (such as the L_2-norm), if exact solution is available.
- Store data in VTK format.
- Display results using visualization software such as VisIt or Paraview.

We implemented all our codes in the Python programming language. For the IGFEM analysis we utilized the Numpy and Scipy packages available in Python. For the 2D and 3D meshing algorithms we used many image processing techniques from the Imaging Toolkit library (ITK [3]).

1.8 Thesis Organization

The remainder of this thesis is organized as follows. In Chapter 2 we discuss 2D meshing, the techniques used in creating the Finite Element mesh, in detecting material interfaces and their intersections with mesh elements, and then we show results. Both polynomial interpolation and non-uniform rational B-spline interpolation are presented in this chapter. In Chapter 3 we discuss the 2D finite element analysis by first introducing the Interface-based Generalized Finite Element method, the types of elements used and their order, how we integrate over these elements, and how we deal with hanging nodes. Then we present results for a thermal problem applied to various domains defined by 2D images. Chapter 4 illustrates the 3D meshing algorithm, in which the techniques and mesh constraints from 2D meshing are extended to three dimensions. We apply this meshing algorithm with planar interpolation and NURBS interpolation to 3D voxel data sets and present the results. Chapter 5 addresses future work and final conclusions.

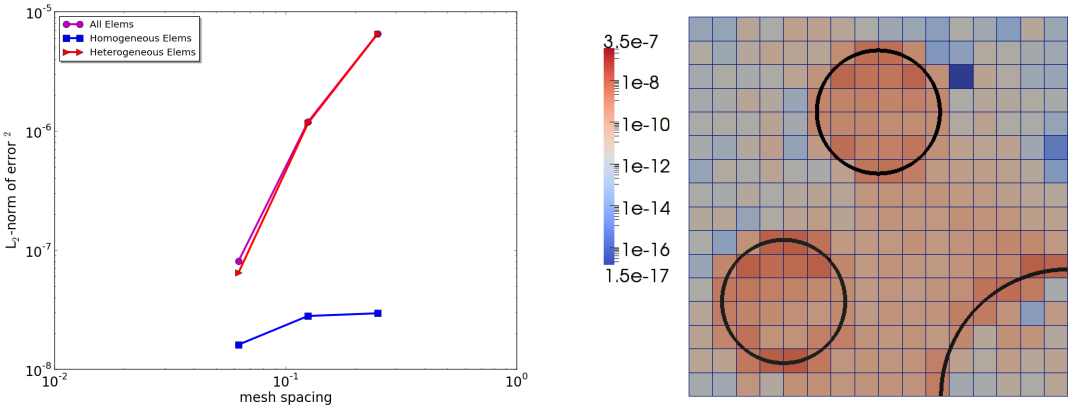
CHAPTER 2

2D Meshing Based on Image Data

In this chapter we present a meshing methodology to be used in finite element analysis of composite materials based on tomographic images of their internal structure. This methodology is governed by requirements of the finite element method, specifically the Interface-based Generalized Finite Element method, or IGFEM, presented in Section 3.3.

When dealing with steep gradients, singularities, or discontinuities in the solution, the mesh used for the numerical solution must capture these accurately. This is usually done by adaptively refining the mesh in areas where the solution error is large. As Figure 2.1a shows, the error associated with heterogeneous elements (i.e., those that intersect a material interface) dominates the overall error of the finite element solution, while homogeneous elements make a relatively small contribution to the overall error. Figure 2.1b shows the L_2 norm of the error in each element of a uniform quadrilateral mesh when linear polynomials are used to approximate the material interfaces. The elements on or near an interface have a greater error than those farther away, which suggests that selective refinement of the mesh near interfaces, as well as more accurate approximation of interfaces would be beneficial in producing a more accurate solution. The approximate error was computed by comparison with a reference solution resulting from a highly refined conforming triangular mesh. More details are provided later in the thesis.

An alternative approach to solution-driven adaptivity is to exploit geometric adaptivity to generate a mesh that accurately captures the complex problem geometry prior to computing the solution, and then employ a solution methodology that takes advantage of this geometric representation, which is the approach we take in this thesis. The most advanced techniques to date are based on selective refinement of the elements. One common approach is recursive decomposition, provided in 2D by quadtree-based algorithms, and in 3D by its three-dimensional generalization, octrees. Quadtrees were first used in the context of finite



(a) (b)

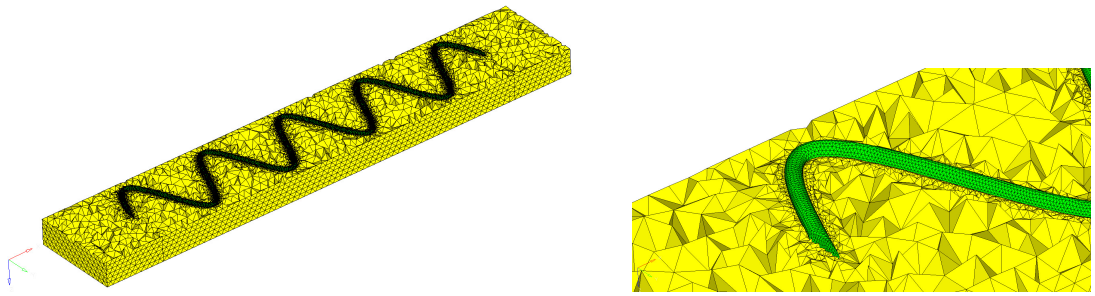
Figure 2.1: (a) Solution error decomposed into elements with interface (heterogeneous) and elements without (homogeneous). (b) L_2 norm of error in each element.

elements by Yerry and Sheppard [57]. We discuss quadtree meshing in this chapter, and 3D octree meshing in Chapter 4.

The remainder of this chapter is organized as follows. Section 2.1 introduces properties of meshes, and Section 2.2 reviews the current state of the art in creating meshes for the finite element method. In Section 2.3 we introduce quadtrees and state the meshing algorithm. In Section 2.4 we discuss how the quadtree data structure can be efficiently accessed in order to locate elements based on index address, and neighbors of elements based on current index location and direction of search. The imaging data used for our experiments is presented in Section 2.5. The mesh constraints imposed by IGFEM are presented in Section 2.6. We describe how the image is segmented to determine the interface and its continuous approximation in Section 2.7.

2.1 Meshes

Meshes are common in the numerical solution of partial differential equations arising from physical phenomena. A poor quality mesh can hinder the ability to analyze the material structure, either by causing inaccurate solutions or by slowing convergence. Hence, one must consider the advantages and disadvantages of using different types of meshes, such as



(a) (b)

Figure 2.2: Highly refined and conforming mesh of a microchannel embedded in a composite material [49].

structured or unstructured, conforming or non-conforming, and different types of elements such as triangles or quadrilaterals in 2D, and tetrahedra or hexahedra in 3D.

Meshes can be categorized based on the connectivity of their interior vertices: structured or unstructured. Structured meshes have all interior vertices topologically alike. Unstructured meshes have vertices with arbitrarily varying connectivity. While unstructured meshes offer more convenient mesh adaptivity/refinement and a better fit to complicated domains, structured meshes offer greater simplicity and efficiency. High-quality hybrid meshes enjoy the advantages of both approaches, but hybrid meshing is not a fully automated process, as it requires user guidance in the decomposition step.

A common approach for 2D structured meshes is to use quadrilaterals, while for 2D unstructured meshes is to use triangles. In the context of tomographic data, quadrilaterals are the natural choice: pixel-aligned meshes and pixel-defined interfaces allow for the finite element mesh to be a subset of the underlying pixel mesh.

The element shape can also have a pronounced effect on the numerical method. Elements that have a large aspect ratio, which is defined as ratio of the maximum to minimum element width, are usually undesirable [16]. Poor aspect ratio affects the condition number of the stiffness matrix in the finite element method and may yield unacceptable interpolation error. The aspect ratio of the element is directly correlated to the internal angles the edges make.

The traditional approach in solving physical problems with discontinuities in the gradient or displacement fields is to use meshes that conform to the geometry of the surface of discontinuity. Creating such meshes that not only accurately represent the actual structure

but also yield elements with acceptable aspect ratio is a complex and expensive process.

In the conventional finite element method, meshes are constructed such that edges in 2D (or faces in 3D) of elements coincide with interfaces between materials or with crack surfaces. Such conforming meshes tend to be not only difficult to generate, but also exceedingly large, and thus computationally expensive. Figure 2.2 shows such a highly refined and conforming mesh of a microchannel embedded in a composite material. The microchannel (meshed with green elements) uses extremely refined elements to represent its geometry. This limitation of classical finite element methods can be overcome by using non-conforming meshes.

2.2 Available Meshing Software for Tomographic Data

Commercial software suites such as Amira [1] and Simpleware [4] have the capability to perform surface reconstruction from micro-CT data and then export a finite element mesh based on a tomographic image. Building a finite element mesh includes segmentation of the tomographic image into separate phases in the material, geometrical definition of the solid boundaries by polygonal facets, and then generation of an unstructured mesh for the solid volume. The resulting finite element mesh conforms to the material interfaces (Figure 2.3) and can be used with standard finite element methods, but it is typically much larger than necessary for a given accuracy.

One of the techniques used by such software to generate volumetric meshes from surface meshes is advancing front algorithms. Advancing front generators are quite popular in aerodynamics simulations [45, 30]. The essential idea behind them is the following: starting from a discretization of the boundary, construct elements from these boundary elements by either connecting two adjacent boundary nodes to an inserted interior node, or by joining two boundary nodes. The process is then repeated as it proceeds inward, advancing like a front, and thus the name [19]. The major drawback of this type of algorithm is that it is computationally expensive and often leads to volumetric meshes of low quality with poor element aspect ratios for complex microstructures [59]. They also do not lend themselves well to adaptivity or to multi-resolution analysis, since they use a fixed, conforming (typically tetrahedral) mesh generated in advance.

A less sophisticated approach to create finite element models relies on the direct translation of voxels to a structured grid of hexahedral elements [35]. Such an approach has been applied to the material shown in Figure 1.1a. The lower-resolution microstructure created by sampling every fouth pixel in the $x-y$ planes is shown in Figure 2.4. While this approach is attractive in its simplicity, it tends to create jagged representations of material interfaces that degrade the solution unless an extremely refined (and computationally expensive) finite element model is employed. Boyd and Muller [17] propose smoothing the low resolution mesh, but this may suffer from loss of fidelity to grain boundary morphology (as some information is lost through smoothing).

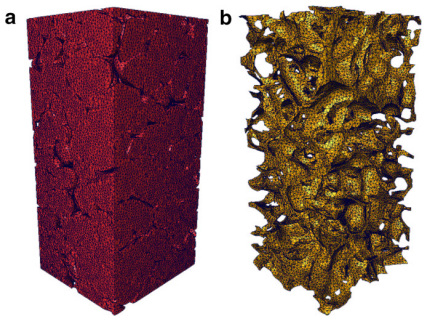


Figure 2.3: Unstructured mesh representation obtained with Amira of fused-cast refractory with zirconia grains (a) and amorphous phase (b) [37].

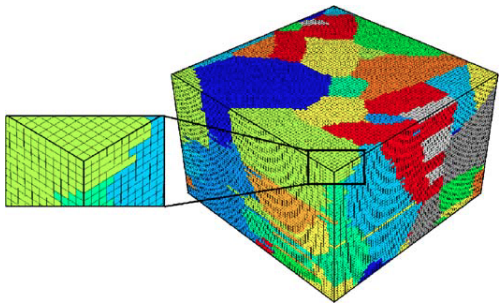


Figure 2.4: Structured mesh representation of polycrystalline material show in Figure 1.1a, based on its 3D tomographic representation. [35].

In this thesis we propose to put the complexity into the method rather than the mesh by using a more sophisticated finite element method with a structured and non-conforming mesh.

2.3 Quadtree Algorithm

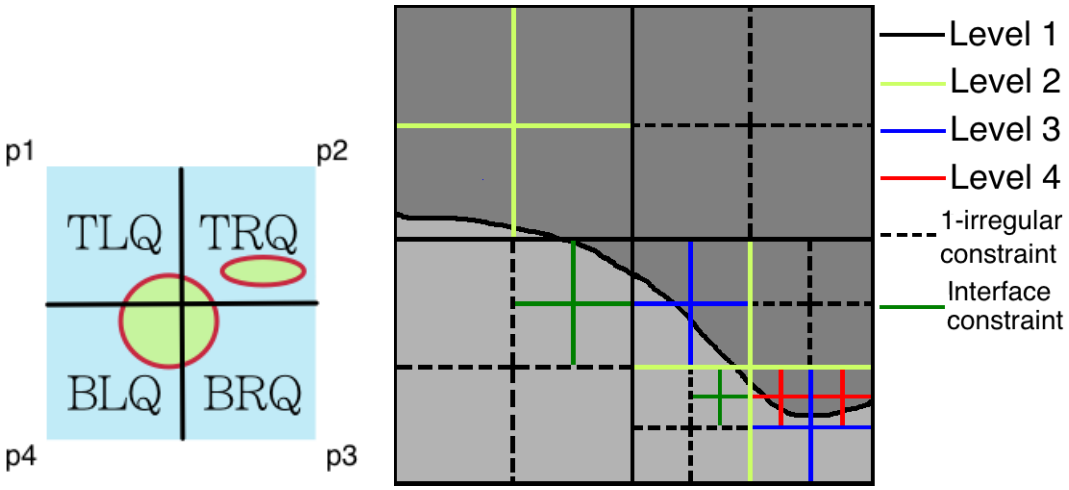
We employ a decomposition method of the domain - quadtree (in 2D) or octree (3D), which divides the data space in a regular way, into rectangles or brick elements. Rectangles and brick elements have the advantage of allowing us to align element edges with the underlying pixel or voxel data and the image coordinate system. The most important parameter in applying this regular scheme is the cell size, which controls the discretization of the domain.

The basic idea behind the general quadtree algorithm [15, 57] can be described in the following way. We start by enclosing the entire 2D domain inside a single axis-aligned rectangle. This root domain is then split into four congruent rectangles, and splitting each of these offspring rectangles into other rectangles continues recursively until a prescribed spatial resolution is obtained. Any additional splits can be dictated either by user-defined requirements or by balance conditions. An example of a balance condition is 1-irregularity: no element should be adjacent to another less than one-half its size. Such a constraint may cause further splits to propagate across the quadtree. The corners of the newly formed elements are called *hanging nodes* if they happen to lie on the edges (and not corners) of the adjacent elements. In the case of a conforming mesh, the quadtree squares would then be warped and cut to adhere to the boundary. A final triangulation of the elements would lead to the creation of an unstructured triangular mesh [16].

Efficient storage and fast data retrievel for quadtrees and octrees make them an attractive technique for generating finite element meshes. Our meshing algorithm is based on a quadtree generator. We recursively decompose the domain into quad elements aligned with pixel boundaries. We selectively refine it near material interfaces inferred from the image and represented by polynomials of desired degree. We discuss this further in Section 2.7.

The selective refinement is governed both by the element size and by the curvature of the interface. The element size is important in attaining the desired accuracy in the subsequent finite element analysis. The maximum element size is dictated by the analysis, while the minimum element size depends on the resolution of the image and the internal geometry of the material.

Our meshing procedure is summarized in Algorithm 1. Starting with an image, a rectangular domain is created corresponding to the size of the image. Using the pixel data from the image and the four corners of the domain, we split the domain recursively into four quadrants: top-left (TLQ), top-right (TRQ), bottom-left (BLQ) and bottom-right (BRQ), as shown in Figure 2.5a. The recursion stops only when the spacing resolution (set by the user by choosing the maximum and minimum element size) and any balance conditions are satisfied. In this case, the balance conditions correspond to the number of interfaces allowed per element, the number of intersections of an interface with an element, and the degree of



(a) (b)

Figure 2.5: Illustrating the meshing algorithm. (a) Four quadrants. (b) Different refinement levels.

the polynomial required to represent the interface accurately. Imposing these constraints concomitantly results in "hq-refinement", where h is the mesh spacing, and q is degree of the polynomials approximating interfaces. The following sections of this chapter discuss this and other constraints.

Figure 2.5b illustrates different levels of refinement applied to an image with an interface. Level 1 and Level 4 correspond to the constraint imposed on the element by the allowed maximum and minimum size, respectively. Level 2 and Level 3 represent refinement triggered when the interface cannot be approximated well by a linear polynomial. An additional level of refinement is triggered by another mesh constraint we call the *1-irregular constraint* or *tree rebalance* (explained below) and which causes propagation of further splits at the parent level. This in turn may cause further splits at its parent, and so on. By *interface constraint*, we mean that no element can have both a hanging node and an interface node.

Each element is allowed only one interface. Any degenerate case that has multiple interfaces inside an element, or an interface crossing an edge multiple times, as portrayed in Figure 2.6, is not permitted, and thus triggers further refinement.

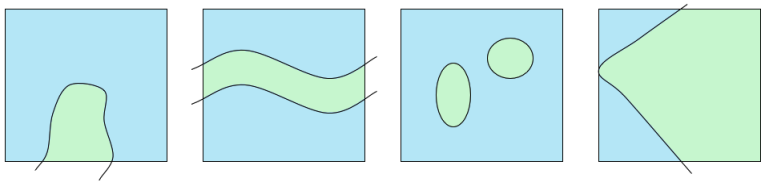
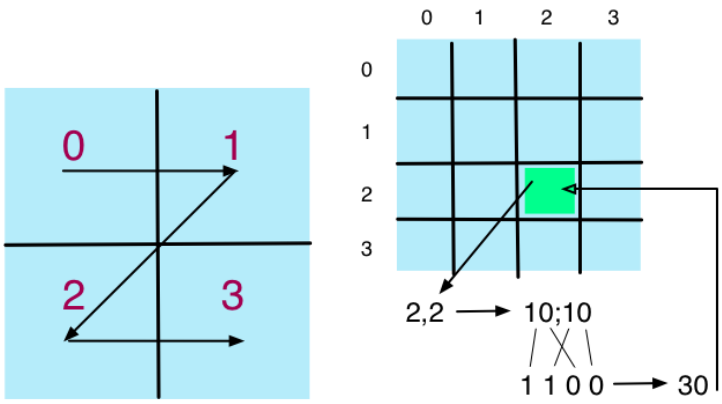


Figure 2.6: We require that an element can have at most two intersection points with an interface, and no inclusions are allowed inside elements. This figure shows a few such anomalies for a heterogeneous element.

Algorithm 1 Generates mesh based on tomographic image

```
1: procedure QUADIFY(image, corners)
2:     if (element size ≥ MAX) then
3:         QUADIFY(image, TLQ corners)
4:         QUADIFY(image, TRQ corners)
5:         QUADIFY(image, BLQ corners)
6:         QUADIFY(image, BRQ corners)
7:     else
8:         if ( in_same_bin (all four corners) ) then
9:             if (large_inclusion_present) then
10:                QUADIFY(image, TLQ corners)
11:                QUADIFY(image, TRQ corners)
12:                QUADIFY(image, BLQ corners)
13:                QUADIFY(image, BRQ corners)
14:            end if
15:        else
16:            for each element-interface intersection case do
17:                approximate interface with polynomial of desired degree
18:                if (approximation "not good enough" & element size ≥ MIN) then
19:                    QUADIFY(image, TLQ corners)
20:                    QUADIFY(image, TRQ corners)
21:                    QUADIFY(image, BLQ corners)
22:                    QUADIFY(image, BRQ corners)
23:                end if
24:            end for
25:        end if
26:    end if
27: end procedure
```



(a) Quadtree partition with index location.

(b) Bit interleaving process.

Figure 2.7: Z-order curve or Morton sequence: a pattern of linear quadtree numbering sequence.

2.4 Indexing and Computing Neighbors

Since quadtrees are a partition of space into four quadrants (in two dimensions), a spatial indexing is required in order to represent each of the cells of the resulting grid. In our algorithm we use a linear quadtree numbering sequence such as the Z-order curve or the Morton curve. Such a space-filling curve preserves locality of the data not just in space, but also when stored in memory. It was first used for spatial indexing by Morton in 1966 [42]. Figure 2.7a shows the Z-order curve.

We use the Morton numbering sequence (as described in [22]) to compute the location codes or indices of all cells in the quadtree. Using data retrieval techniques such as the Finite-State-Machine algorithm [58], we can then easily identify not just the spatial location of a given cell, but also its neighbors and information about them. The first step in going from a quadrant with only coordinate information to creating an indexing for it, is to assume a uniform partition of the domain in both x and y directions, and then obtain the row and column location of the cell in this uniformly partitioned domain. Once we have these indices (i, j), we convert them to binary, do bit interleaving, and then convert the result to base 4 to obtain the index of the cell. Figure 2.7b shows the bit interleaving process graphically.

Consider the example shown in Figure 2.8a. We identify the cell in row 2 and column 5

and would like to compute the index location for it:

$$2_{\text{base 10}} = 10_{\text{base 2}},$$

$$5_{\text{base 10}} = 101_{\text{base 2.}}$$

After bit interleaving the index address is

$$11001_{\text{base2}} = 25_{\text{base10}} = 121_{\text{base 4}}.$$

Mathematically, we can express the bit-interleaving by a one-to-one function. Given the pair of coordinates in base 10 (X_{10}, Y_{10}), converted in base 2 to (X_2, Y_2), the Morton number is given by

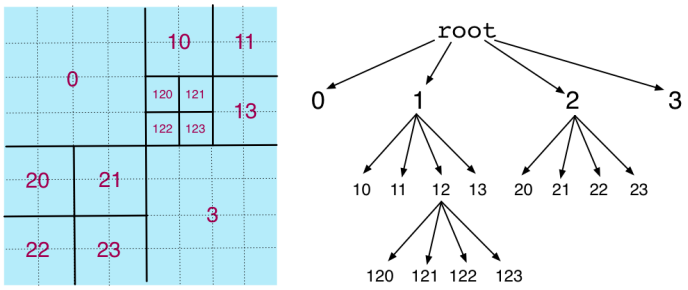
$$M = \frac{\sum_{i=1}^{T}(2^{2i} * x_{i-1} + 2^{2i+1} * y_{i-1})}{2^2},$$

where T is the maximum length of the string representation of X_2 and Y_2, and x_i and y_i represent the bit indices of X_2 and Y_2, respectively.

In our example, $(5, 2)_{10} = (101, 10)_2$, thus

$$M = \frac{(2^2 * 1 + 2^3 * 0) + (2^4 * 0 + 2^5 * 1) + (2^6 * 1 + 2^7 * 0)}{2^2} = \frac{100}{4} = 25_{10} = 121_4.$$

The indexing or location code for each child in the tree describes a unique path from the root to the node. Determining the neighbors of a given node is then a matter of finding the location code for these neighbors. Using location codes and directions, we can determine all surrounding neighbors by using the Finite-State-Machine (FSM) algorithm described in [58]. The basic idea behind this algorithm is to compute the index of the neighboring node in the desired direction, followed by a search in the quadtree for the existence of this node. In the case of 1-irregular trees, if the node does not exist it could be that the neighbor is at a depth level that is either one higher or one lower than that of the node for whose neighbors we are probing. The computation of the location code of the neighbor involves several lookups through Table 1 (also shown below) [58] . In the worst case, the FSM algorithm does n table



(a) (b)

Figure 2.8: (a) Quadtree partition with index location. (b) Quadtree representation of 2.8a.

lookups, where n is the depth of the target node and the first common ancestor is the root node. FSM also requires n lookups when the node does not exist in the tree. In the best case, when the nodes are siblings, it takes only one table lookup.

Direction	Quadrant 0	Quadrant 1	Quadrant 2	Quadrant 3
R	1, H	0, R	3, H	2, R
L	1, L	0, H	3, L	2, H
D	2, H	3, H	0, D	1, D
U	2, U	3, U	0, H	1, H
RU	3, U	2, RU	1, H	0, R
RD	3, H	2, R	1, D	0, RD
LD	3, L	2, H	1, LD	0, D
LU	3, LU	2, U	1, L	0, H

Table 2.1: Finite-State-Machine lookup table for quadtree neighbors [58]. L - for left, R - for right, D - for down, U - for up, H - for halt.

For example, for the quadtree illustrated in Figure 2.8, the left-up neighbor of 120 would be computed using the following three steps, since the length of the location code is 3.

1. Starting with the right-most index 0, look in the table in column labeled Quadrant 0 and row labeled LU for left-up neighbor. The result is $3, LU$, meaning 0 is replaced by 3 and the new direction becomes LU. New index is 123.

2. Next, for second right-most index 2, using the new direction LU look in the table in column Quadrant 2 and row LU. The output is $1, L$. New index is 113 and the new direction is L.

3. For the last index, 1, we use the new direction L and we obtain for Quadrant 1, direction L, $0, halt$. 1 is replaced by 0 to become: 013. The new direction is $halt$, thus no more lookups in the table are necessary.

The left-up neighbor of 120 is 013. A search in the quadtree shows that this neighbor does not exist. Chopping off the last index 3, the next search in the quadtree is done for node with location code 01. Since this node does not exist, then 0 is looked up and found.

Test for homogeneity

With each element of the quadtree, we associate a property of homogeneity that we determine recursively. Starting from the maximum size element, we first determine whether the element is homogeneous or heterogeneous. An element is *homogeneous* if all the underlying pixels represent the same phase in a material, and *heterogeneous* otherwise. We determine whether an element is homogeneous by sampling the pixel values it contains. Each material has a range of pixel values attributed to it. We initially check the four corners of the element, which often suffices to detect heterogeneity at low cost. If an element passes this test, then we conduct a more rigorous search using random sampling. For an element to be truly homogeneous, it should contain no inclusion of significant size within it. For example, for a 64×64 pixel element, any inclusion that is more than 4×4 pixels might be considered significant. We derived a formula to compute the number of random samples required to achieve a given probability that no significant inclusion has been missed after a chosen number of uniform random trials.

The probability of missing a heterogeneous inclusion of size $n \times n$ pixels, after k uniform random trials, in a $m \times m$ pixels domain is given by:

$$P = \Big(\frac{m^2 - n^2}{m^2}\Big)^k.$$

From this formula, we can compute the value of n for a given P and k.

Table 2.2 shows a few values we have considered in our meshing implementation. For relatively large elements compared to inclusion areas (a 64×64 pixels element with an

inclusion of 4×4 pixels), we would sample at most 14.4% of the total points to obtain a probability of only 10% (i.e., a 10% chance) that a significant inclusion could have eluded detection, or a probability of 90% (i.e., a 90% chance) that a significant inclusion would have been detected, assuming there is one. The percentage of total points sampled drops even more for larger inclusions or lower probabilities that we detected an inclusion, as can be seen from the table: 3.6% of the total points for a probability of 10% that we missed an 8×8 pixels inclusion in an 64×64 pixels element. As seen from the table, random sampling is much faster in determining whether there are inclusions inside an element, versus doing a pixel by pixel search of the whole element. A nice feature of random sampling is that its complexity does not depend on the dimension.

Element Area $m \times m$	Inclusion Area $n \times n$	Probability Missed Inclusion P	Number of Samples k	Percentage of Total Points Sampled
$64 \times 64 =$ 4096 px	4×4	10%	589	14.4%
		30%	307	7.5%
	8×8	10%	147	3.6%
		30%	77	1.9%
$32 \times 32 =$ 1024 px	4×4	10%	147	14.4%
		30%	77	7.5%
	8×8	10%	36	3.6%
		30%	19	1.9%

Table 2.2: Statistics for random sampling for homogeneity. All data reported is per element.

Element-Interface Intersection

Another key constraint associated with the mesh generation process involves limiting the number of interfaces present in a given quadrilateral element and resolving the mesh to capture the curvature accurately. These requirements can be enforced by not allowing multiple interface-edge intersections, and by allowing in an element at most two edges to be crossed by an interface. This requirement is needed in order to simplify the finite element analysis utilizing the mesh. The mesh elements are then one of 17 types: homogeneous (1), or heterogeneous (16) with an interface passing through them as illustrated in Figure 2.9. In effect, the 16 heterogeneous types of elements can be reduced to four: one case splits the

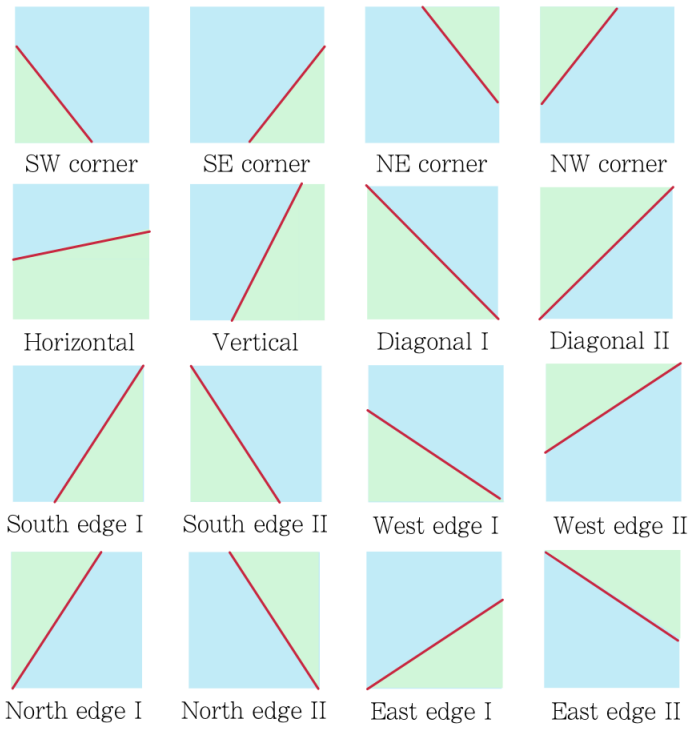


Figure 2.9: (2D) Various cases in which an interface can intersect a quad element.

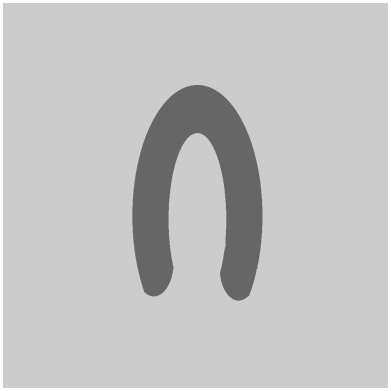
domain into a triangle and a pentagon, another into two quadrilaterals, another into two triangles, and yet another into a triangle and a quad. By allowing only these four types of intersection, we simplify the geometrical representation of the interfaces, as well as allow for a simpler and more straightforward implementation of our numerical method used to perform the finite element analysis of the material. The numerical method is discussed in Chapter 3.

2.5 Test Images

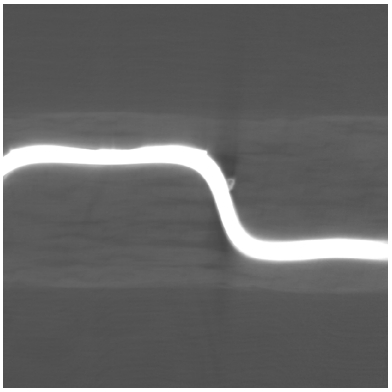
The remaining sections of this chapter will introduce the meshing procedure and discuss various features of the methodology. To illustrate some the points we use a series of images shown in Figure 2.10 that are either artificially created or acquired through X-ray tomography. These images will be used selectively to illustrate various concepts and issues. Here is a summary of these images.

(a) Circles

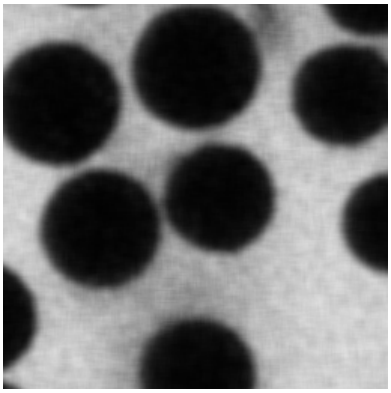
(b) Channels



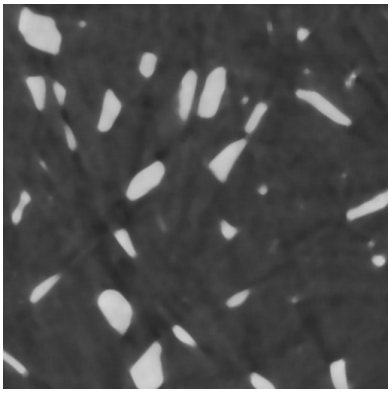
(c) Horseshoe



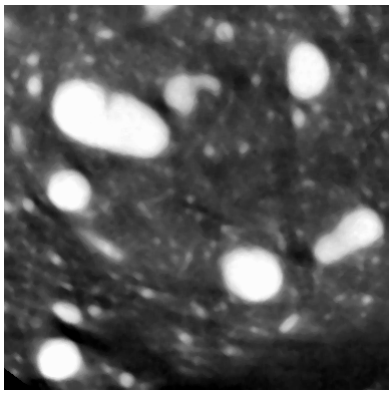
(d) Microvascular



(e) Fibers



(f) SiBat



(g) SnBat

Figure 2.10: Images considered for computational experiments.

- Circles (Figure 2.10a) is an artificially created image. Its size is 1000 $\times$ 1000 pixels. The two circles have a radius of about 166 pixels, while the quarter circle has a radius of 333 pixels. Circular inclusions are found in many composite materials, making this image relevant to real life materials.

- Channels (Figure 2.10b) is an artificially created image of size 1000 $\times$ 1000 pixels. The width of each channel varies from about 50 pixels to about 100 pixels. This geometry is inspired by microvascular materials.

- Horseshoe (Figure 2.10c) is an artificially created image of size 1000 $\times$ 1000 pixels. This image is relevant in illustrating an extreme case for the steep-gradient constraint discussed in the next section. The inclusion width is kept relatively constant throughout, at 100 pixels.

- Microvascular (Figure 2.10d) is a tomographic image of an embedded microchannel in a polymer. The image is 700 $\times$ 700 pixels. It contains noise from the X-ray machine, as well as a smudge from the manufacturing process of the material; however, it has very good contrast. The widest part of the channel is about 30 pixels.

- Fibers (Figure 2.10e) is a tomographic image of a reinforced composite. The circular inclusions represent glass fibers embedded in resin. This image is of size 256 $\times$ 256 pixels. In its raw format, this image had a lot of noise that made it difficult to distinguish the fiber boundaries. To reduce the effect of noise, we applied a smoothing algorithm that averaged the pixel values over a local area.

- SiBat and SnBat (Figure 2.10f and Figure 2.10g) are tomographic images of silicon and tin materials, respectively, for battery electrodes. Figure 2.10f is of size 512 $\times$ 512 pixels, while Figure 2.10g is of size 384 $\times$ 384 pixels. These images pose various challenges, as the inclusions are quite irregular and quite small. The inclusion size is constrained by image resolution and can directly affect the mesh quality. A smoothing algorithm was also applied to these images in order to delimit better the silicon and tin electrode boundaries.

2.6 IGFEM-Imposed Constraints

The motivating principle of this thesis is to put the complexity into the method, rather than the mesh. Thus, having simpler, smaller meshes is highly desirable. The error in the solution from a finite element method is directly dependent on the element size or spacing h. By uniformly refining the mesh, good solution accuracy can be achieved but at the cost of greater computational resources. The desired accuracy can be achieved without major impact on the overall mesh size and in an economical way by allowing automatic adaptive mesh refinement in regions that require more elements to resolve the solution adequately. The automation is done by presetting various tolerances and employing a recursive algorithm such as the quadtree generator until these tolerances are met. The refinement is done locally at each interface boundary. In classical finite element methods, adaptive meshes conform to the interface, whereas IGFEM meshes do not.

In our research, the mesh adaptivity is driven by the required degree of geometric fidelity, as well as any constraints imposed by the underlying finite element method. Several such constraints are incorporated into our meshing procedure. We will first list these constraints, and then proceed to detail them one by one.

IGFEM imposed constraints:

1. h- and q- refinement,
2. either irregular nodes or enrichment nodes are allowed in a single element, but not both,
3. 1-irregular rule,
4. 3-neighbor rule or tree re-balance,
5. steep-gradient constraint.

Constraints 2, 3, and 4 each require several passes through the same constraint before the next one is applied, since each pass may create new elements that violate the constraint. The constraint is considered correctly applied after the mesh no longer changes with subsequent passes. Applying constraint 5 always leads to a mesh that violates constraints 2, 3 and 4.

Thus, these must be applied again in the same iterative fashion. We consider the mesh to be finalized only after any additional pass no longer changes the mesh. At the end of this section we will quantify how the various constraints affect the mesh size for several of our images.

2.6.1 hq-refinement

Domain discretization is a necessary step in finite element computations, and the resulting discretization error depends on the element size, h. Taking h to be smaller is known as h-refinement. In the context of IGFEM, the error also depends on how well the material interfaces are approximated. The interfaces can be approximated by polynomials of different orders (linear, quadratic, etc.) or with other functions such as B-splines. We denote the order of such an approximation by q. Taking q to be larger is called q-refinement. Allowing both h and q to vary is called hq-refinement.

For q-refinement we consider Newton polynomial interpolation of order one ($q = 1$), two ($q = 2$) and three ($q = 3$), and non-linear uniform rational B-splines (NURBS) of order 3. For Newton polynomials we do not consider any other higher order because a cubic often is enough, and because we want to keep a balance between cost of constructing a polynomial and cost of utilizing them in the IGFEM analysis. For B-Splines we consider only quadratic NURBS because they provide good approximations at low costs. These approximation methods are discussed in detail in Section 2.7.

We do h- and q- refinements concomitantly in the following way

- For a fixed h, various values of q are tried. If a tolerance measuring the accuracy of the approximation to the interface is not met, a new, smaller h is chosen and the process is repeated, until the smallest allowed value for h is reached.

Before any q-refinement is tried, for each heterogeneous element, we first detect the intersection points of the interface with the element edges. We call these intersection points *enrichment nodes*, because the finite element solution will be "enriched" at these nodes (see Chapter 3). If more than two of these nodes are detected per element, then h-refinement is

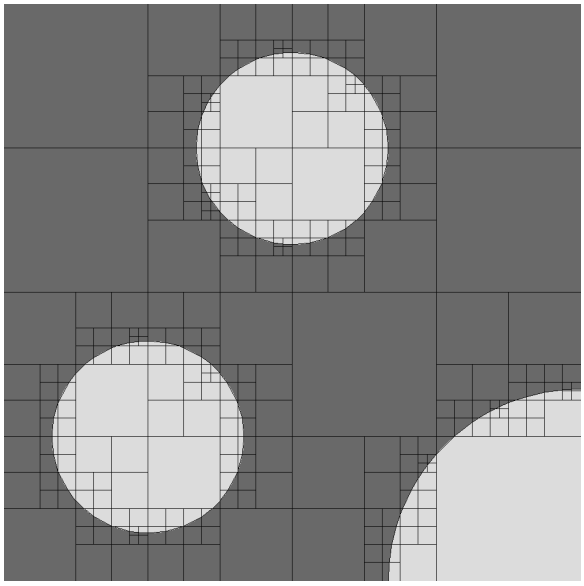


Figure 2.11: Circles: Only hq-refinement was allowed, with $q = 1$.

triggered. This is part of the test for homogeneity discussed earlier in this chapter. Any interior nodes used for higher order approximations will be used only as part of the integration domain when computing the load vector and stiffness matrix. Note that q-refinement does not create curved elements, but can create curved domains of integration. Such domains are transformed through isoparametric mapping, also discussed in Chapter 3.

hq-refinements has several nice properties

1. keeps the mesh as coarse as possible, while still enabling the required solution accuracy,
2. keeps the mesh structured,
3. quadrilaterals do not conform to the geometry of the problem, but their interior interfaces do,
4. no increase in enrichment nodes by using higher order q.

Figures 2.11, 2.12, 2.13 and 2.14 show the mesh with only the hq-refinement constraints applied. These meshes are not yet ready to be passed to the finite element method, as several additional constraints must be applied.

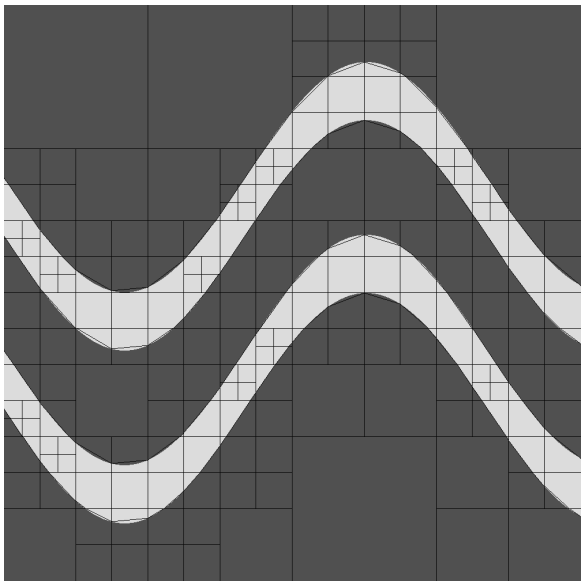


Figure 2.12: Channels: Only hq-refinement was allowed, with $q = 1$.

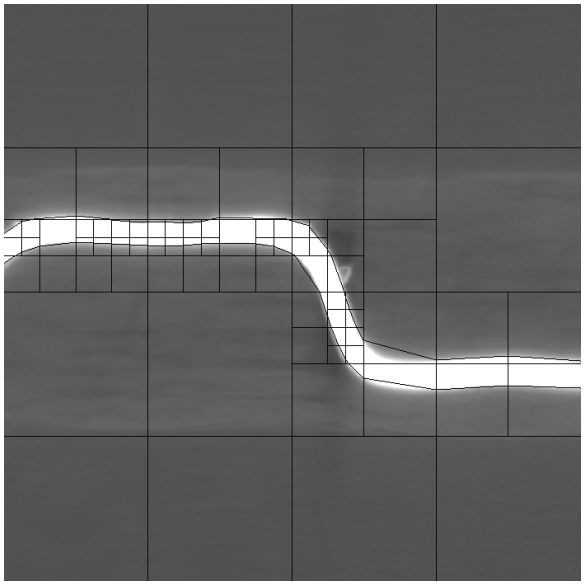


Figure 2.13: Microvascular: Only hq-refinement was allowed, with $q = 1$.

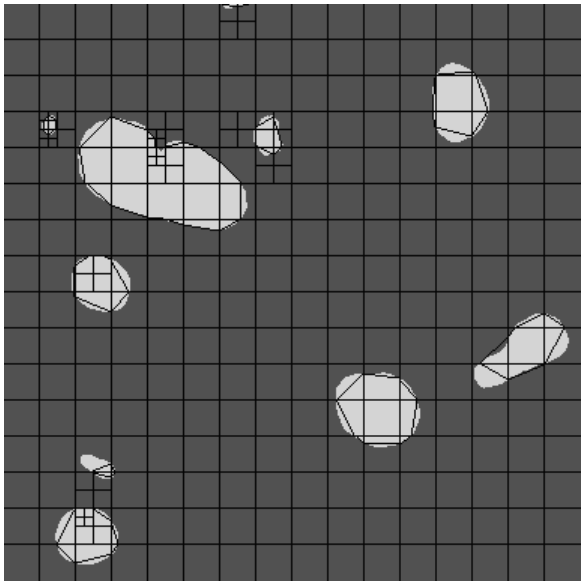


Figure 2.14: SnBat: Only hq-refinement was allowed, with $q = 1$.

2.6.2 Interface Constraint

A *hanging node* or an *irregular node* is defined as a node where the edges of a refined element meet at the midsides of a coarser one. Adaptive refinement leads naturally to the creation of hanging nodes. Some of the IGFEM constraints create them when refinement is triggered locally in only one element, without the surrounding elements being affected by that constraint. An element crossed by an interface creates additional nodes (besides the corner nodes) that the finite element method must treat in its formulation. These are the so called *enrichment* nodes. To avoid a complicated finite element formulation, we require that no element crossed by an interface should have hanging nodes on its edges. This constraint enables a straightforward application of IGFEM in the local element, at the cost of slightly increasing the number of elements in the mesh.

The constraint is applied in the following way. If an element has both a hanging node and one or more enrichment nodes, the element is subdivided. We look at each surrounding neighbor − north, south, east, west − and check whether those elements have children elements. If we find one that does, the subdivision is triggered in that element. This process is repeated several times, through several passes through the mesh, in order to make sure that the newly subdivided elements do not themselves create additional hanging nodes. For

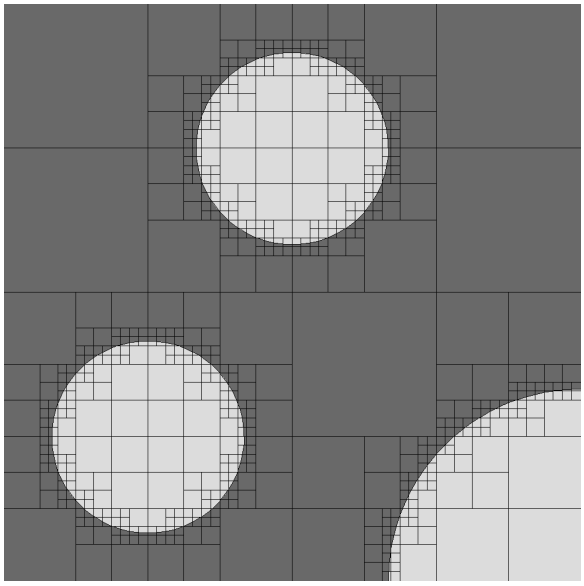


Figure 2.15: Circles: After hq-refinement was applied, we allowed no element to have both hanging nodes and enrichment nodes.

the same four images used in the previous section, we applied this requirement on the mesh after the hq-refinement: Figures 2.15, 2.16, 2.17 and 2.18.

2.6.3 1-irregular Rule

The previous constraint limits the number of hanging nodes a heterogeneous element can have to zero. It does not, however, limit the number of hanging nodes for homogeneous elements. Multiple difficulties arise when there are too many irregular nodes on an edge. To overcome these, we use the approach developed by Bank [12, 11] which uses the "1-irregular" rule. The 1-irregular rule limits the number of hanging nodes an edge can have to one. Any more than this triggers further subdivision at the parent level, which can also trigger further subdivision at its parent level, and so on. This constraint is inherited by IGFEM from traditional finite elements.

The procedure for applying this constraint is similar to the procedure for the previous constraint. For each homogeneous element, we look at its surrounding neighbors: north, south, east, west. If any of them has children that have children, this indicates that there will be more than one irregular node present on that edge. The homogeneous element is subdivided

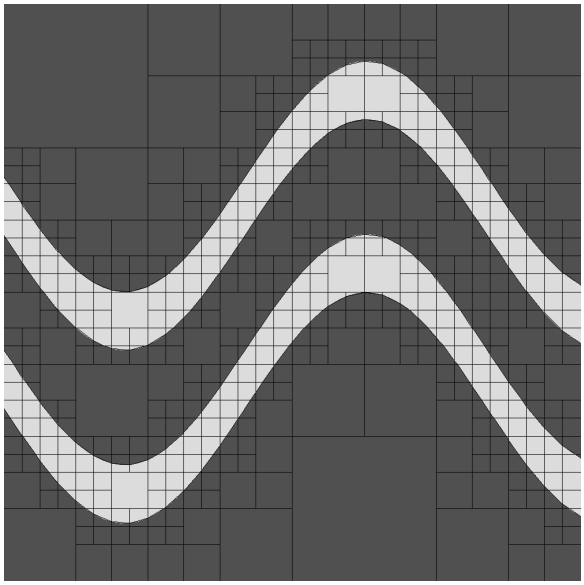


Figure 2.16: Channels: After *hq*-refinement was applied, we allowed no element to have both hanging nodes and enrichment nodes.

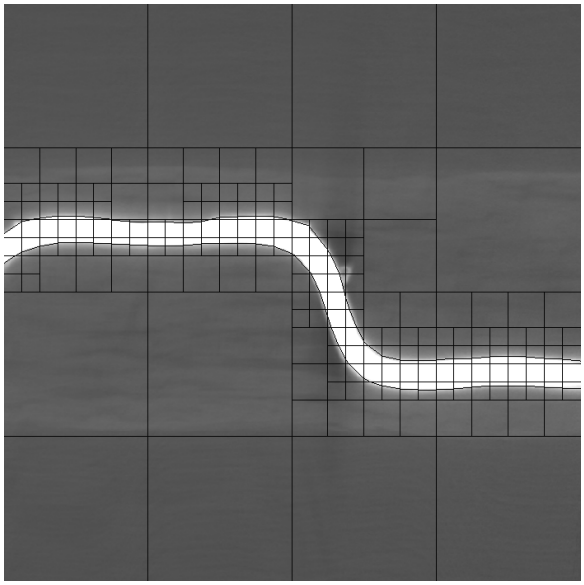


Figure 2.17: Microvascular: After *hq*-refinement was applied, we allowed no element to have both hanging nodes and enrichment nodes.

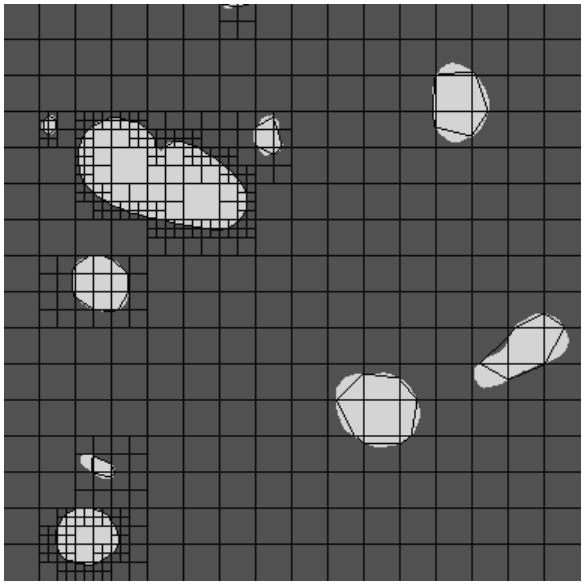


Figure 2.18: SnBat: After *hq*-refinement was applied, we allowed no element to have both hanging nodes and enrichment nodes.

as soon as the first neighbor with grand-children is discovered. Several applications of this constraint are also needed before we can conclude that the 1-irregular requirement has been successfully applied.

Figures 2.19, 2.20, 2.21 and 2.22 show how the mesh changes after this constraint is applied. The increase in number of elements can be large or small, depending on the underlying image geometry.

2.6.4 3-neighbor Rule

Another rule introduced by Bank [12, 11] and used in regular finite element methods is the "3-neighbor" rule. The three-neighbor rule states that any element having irregular nodes on three of its four edges must be refined. In other words, at most two surrounding neighbors can be subdivided. Anything more triggers refinement.

Again, the procedure to apply this constraint is similar to the previous ones: we look at the surrounding neighbors on all sides, and as soon as we find more than two that have children, we subdivide. We need consider only homogeneous elements, as no heterogeneous element is allowed to have both irregular nodes and enrichment nodes.

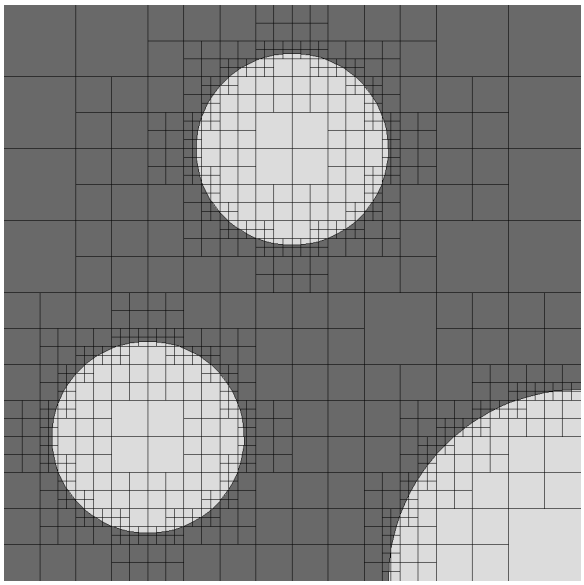


Figure 2.19: Circles: After 1-irregular rule was applied to mesh.

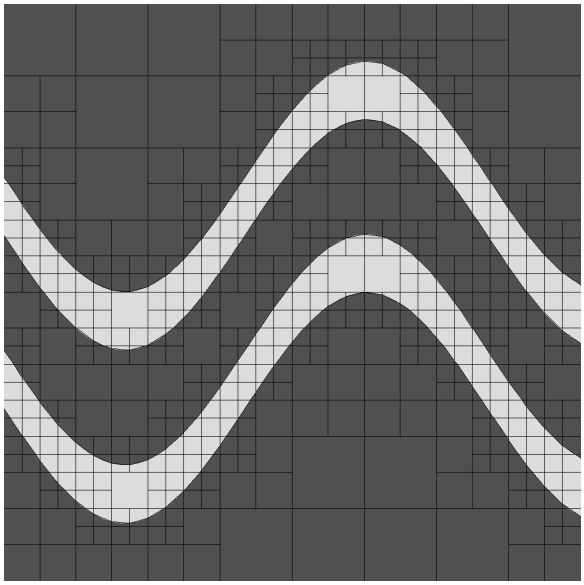


Figure 2.20: Channels: After 1-irregular rule was applied to mesh.

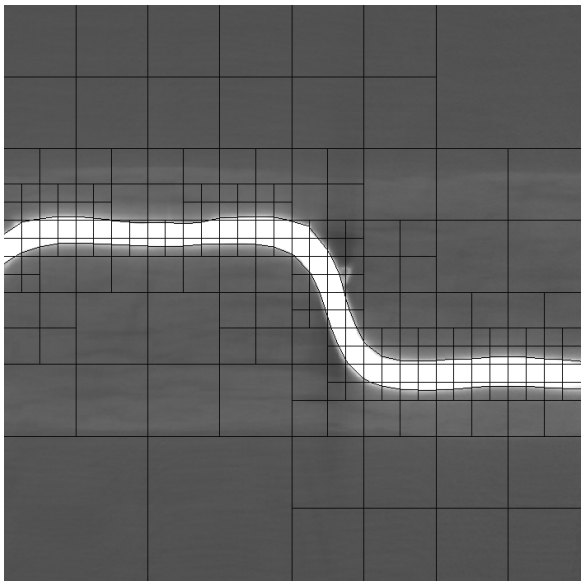


Figure 2.21: Microvascular: After 1-irregular rule was applied to mesh.

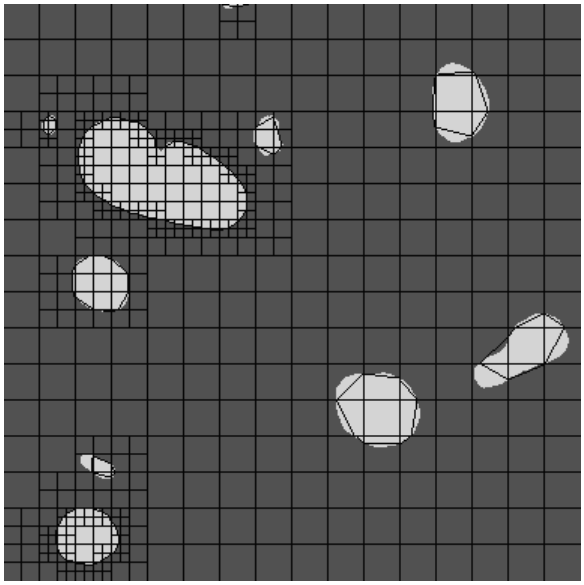


Figure 2.22: SnBat: After 1-irregular rule was applied to mesh.

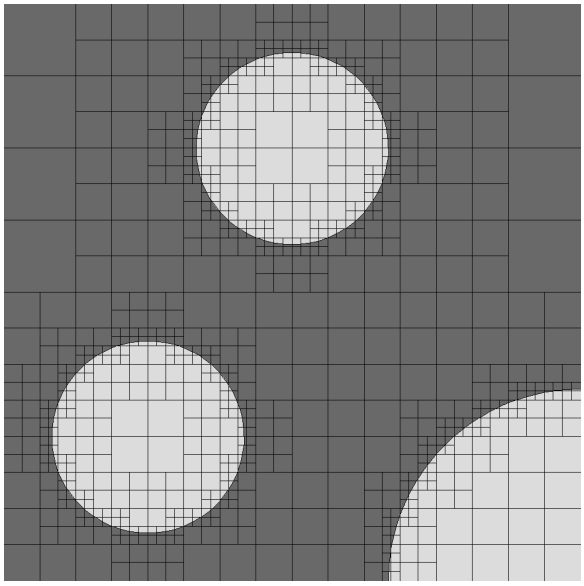


Figure 2.23: Circles.png: After 3-neighbor rule was applied to mesh.

For most of our images this constraint has very little effect on the overall mesh size, in some cases the change being almost insignificant. We can see this in the meshes shown in Figures 2.23, 2.24, 2.25 and 2.26, which hardly change from the previous constraint.

2.6.5 Steep-gradient Constraint

When two inclusions in a material are very close to each other, steep gradients may arise in the solution that may make it difficult to resolve accurately. When this is the case, in order to capture the solution in the inclusions accurately, we must increase the number of elements in the mesh at these locations. Conventionally, these locations are usually identified after a solution is obtained and appraised, in what is known as *a-posteriori error analysis* [5, 47, 56]. This a-posteriori approach, however, does not lend itself well to an automatic adaptive meshing algorithm.

In order to have a meshing algorithm that does not need to be restarted to do "fixes," we developed a procedure that detects potential troublesome regions during the meshing process.

The algorithm presented below makes use of normal vectors to inclusion interfaces:

- consider the line segment connecting the two enrichment nodes determined by an

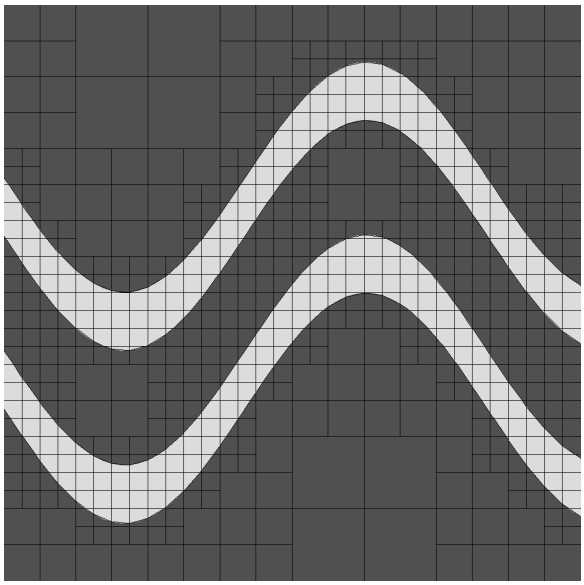


Figure 2.24: Channels.png: After 3-neighbor rule was applied to mesh.

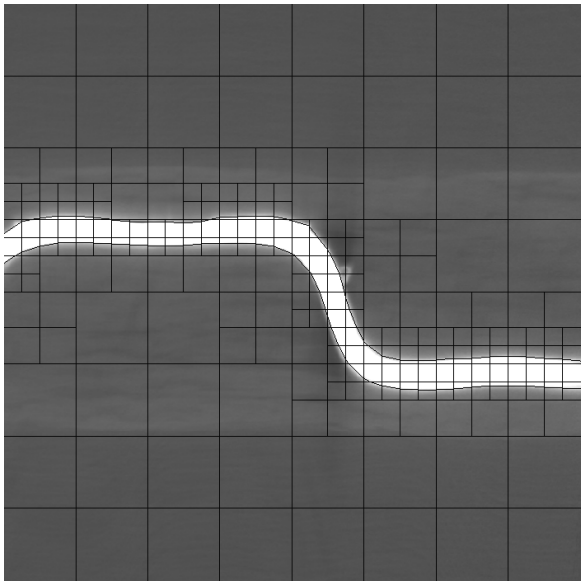


Figure 2.25: Microvascular.png: After 3-neighbor rule was applied to mesh.

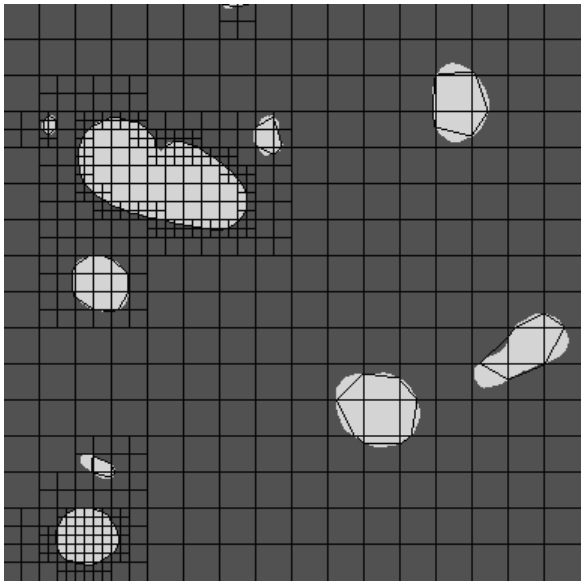


Figure 2.26: SnBat.png: After 3-neighbor rule was applied to mesh.

interface intersecting a given element,

- construct the normal to this line segment at its midpoint,
- along this normal, look in both directions and count the number of homogeneous elements crossed as we proceed along the line until a heterogeneous element is encountered
- if the number of homogeneous elements is smaller than a desired minimum value, refine the intervening homogeneous elements

This procedures uses an alternative minimum element size that can be smaller than the preset minimum, and is used only in steep gradient areas. This alternative minimum size for h-refinement can be as small as 1 pixel (goes down to image resolution). The user has the option of choosing up to ten elements between interfaces. We chose ten as a reasonable number and more than the user may actually need.

For the simple mesh shown in Figure 2.27a, we compute the normal vector to the interface in each heterogeneous element and then search along these normals, as shown in Figure 2.27b, for homogeneous elements crossed when going from one heterogeneous element to another. If either the count is below a user set threshold or a heterogeneous element is encountered before the count is completed, the search is stopped and the traversed homogeneous elements are

subdivided up to the minimum number of homogeneous elements between two interfaces. The search is done in both directions of the normal. In Figure 2.27b we show only the normals along which a search was done. Figure 2.27c shows the mesh after the constraint was applied. As we can see, between the two smaller inclusions there were initially an insufficient number of homogeneous elements, so this constraint triggered refinement. There were also not enough homogeneous elements within the same inclusion, from one interface to another.

We also consider a more extreme case: where the image has an inclusion in the shape of a horseshoe. We show this image and its mesh in Figure 2.28. The normals shown in Figure 2.28b illustrate the direction of search and answer the question of what happens when two interfaces are very close to each other, but part of the same inclusion.

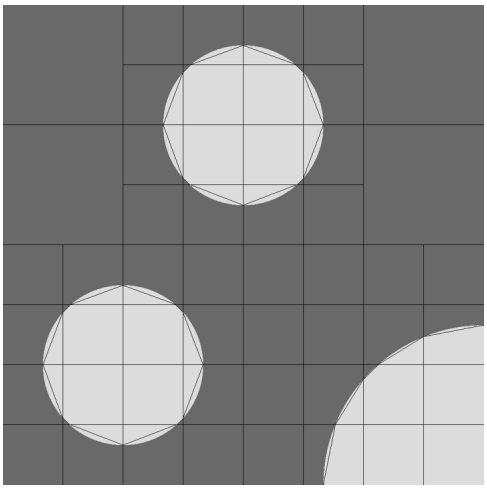
Figures 2.29, 2.30, 2.31 and 2.32 show how the mesh changed when the steep-gradient constraint was triggered.

2.6.6 Final Constraints

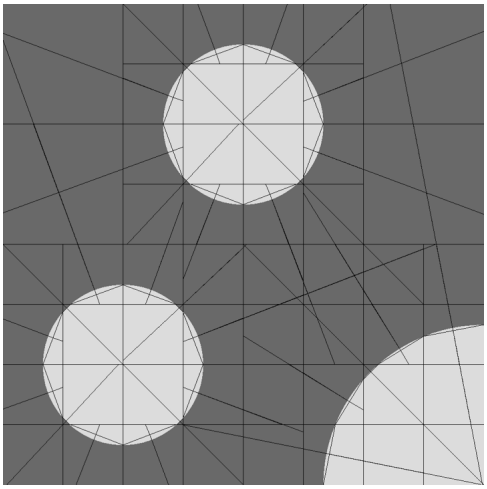
The steep-gradient constraint creates meshes that violate some of the previous constraints, such as the interface constraint, 1-irregular rule, or 3-neighbor rule. To rectify this, a final iterative pass through each of these is needed in order for the mesh to be ready for the finite element method. Figures 2.33, 2.34, 2.35 and 2.36 show the final meshes after all constraints were applied. Table 2.3 shows statistical information about the mesh size for each of these images.

	Circles		Channels		Microvascular		SnBat	
	total elems	CF	total elems	CF	total elems	CF	total elems	CF
hq-refinement	292	1	193	1	76	1	301	1
interface constraint	616	2.1	523	2.7	187	2.5	505	1.7
1-irregular rule	838	2.9	550	2.8	223	2.9	550	1.8
3-neighbor rule	856	2.9	628	3.3	232	3.1	565	1.9
gradient constraint	1036	3.6	2089	10.8	667	8.8	739	2.5
all constraints	1366	4.7	4915	25.5	1570	20.7	886	2.9

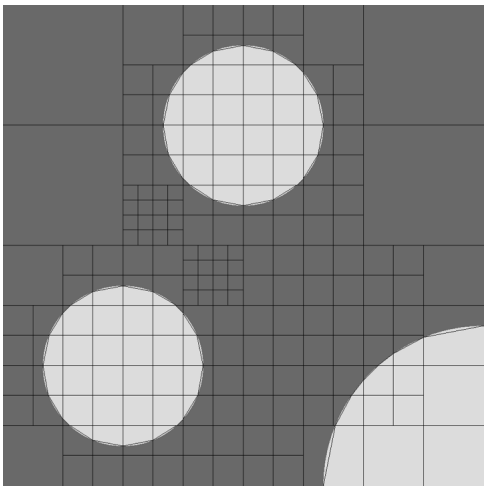
Table 2.3: Statistical information about mesh size and cumulative increase after each major IGFEM constraint was applied. Several images with various geometries were considered.



(a)



(b)



(c)

Figure 2.27: Circles: Simple mesh where steep-gradient constraint is triggered by search for homogeneous elements along normals to interface.

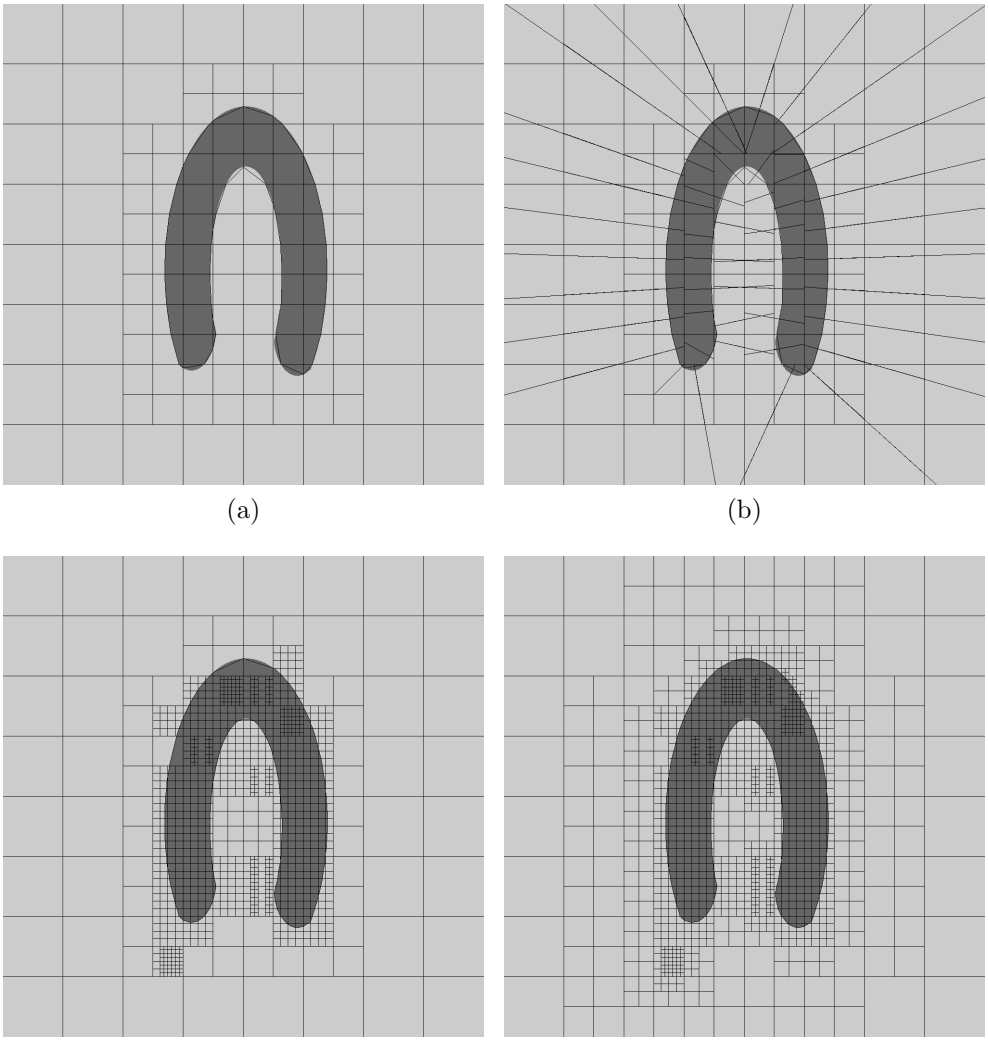


Figure 2.28: Horseshoe: Simple mesh where steep-gradient constraint is triggered by search for homogeneous elements along normals to interface.

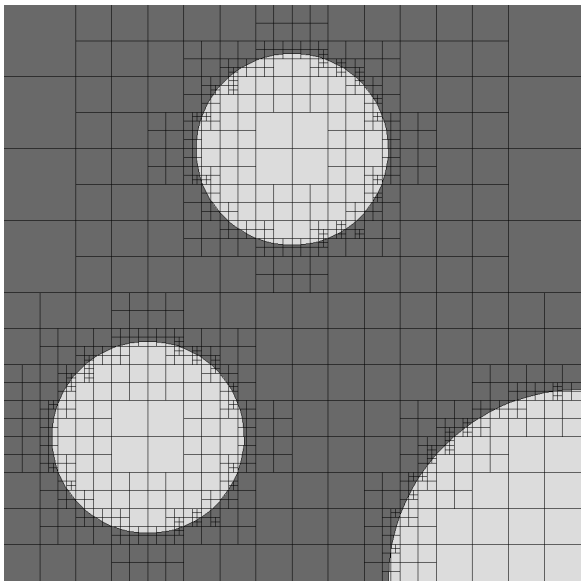


Figure 2.29: Circles: After steep-gradient constraint is applied to mesh.

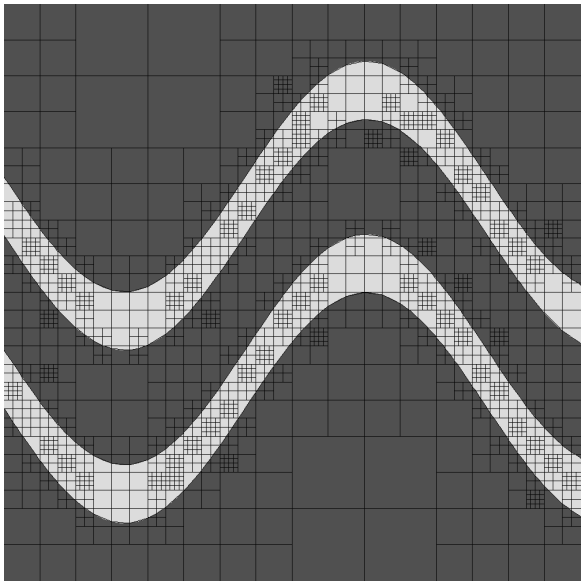


Figure 2.30: Channels: After steep-gradient constraint is applied to mesh.

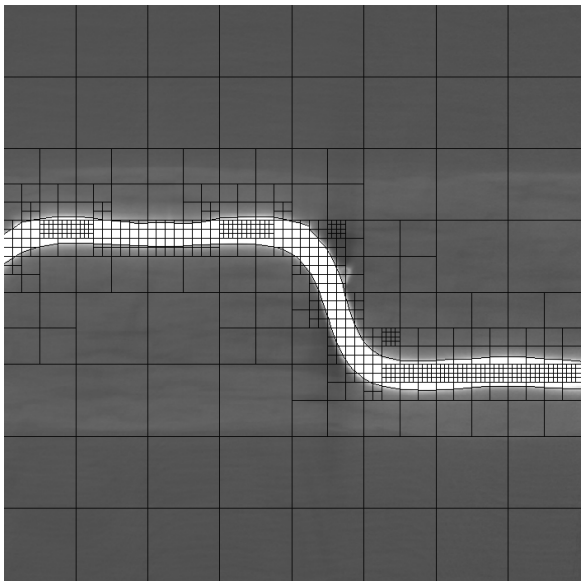


Figure 2.31: Microvascular: After steep-gradient constraint is applied to mesh.

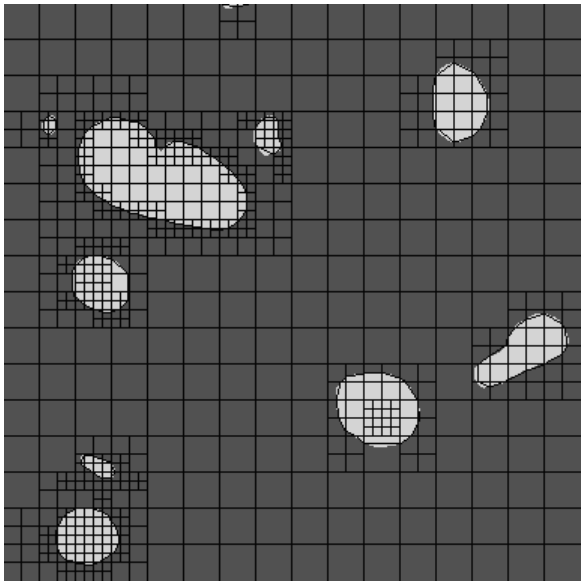


Figure 2.32: SnBat: After steep-gradient constraint is applied to mesh.

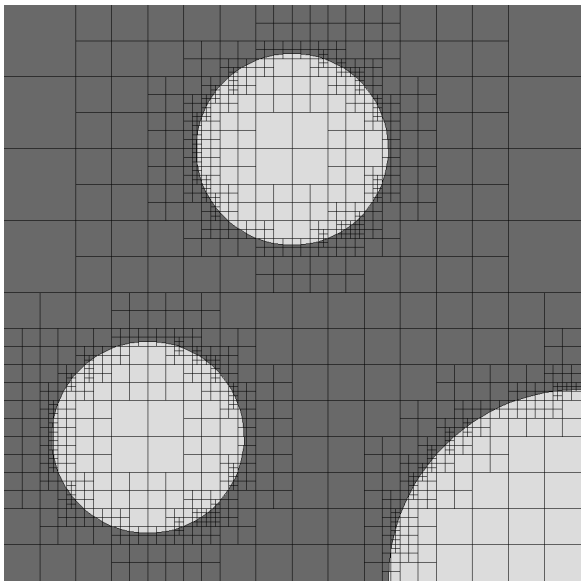


Figure 2.33: Circles: After all IGFEM constraints were applied to mesh.

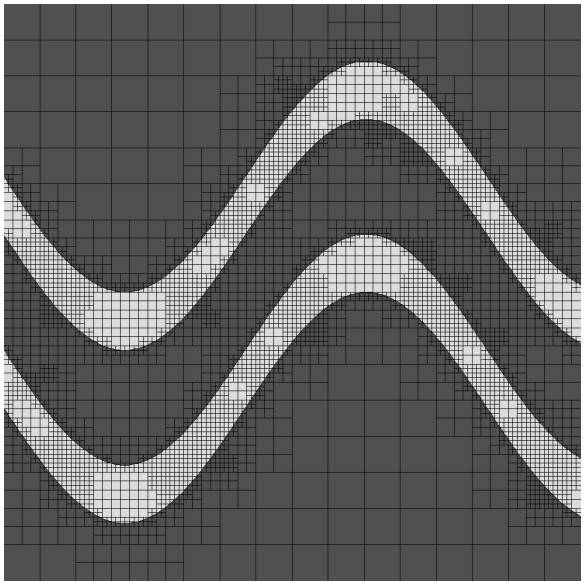


Figure 2.34: Channels: After all IGFEM constraints were applied to mesh.

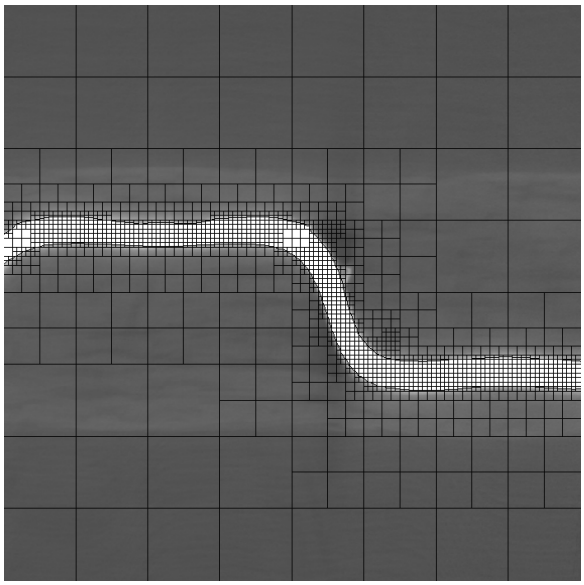


Figure 2.35: Microvascular: After all IGFEM constraints were applied to mesh.

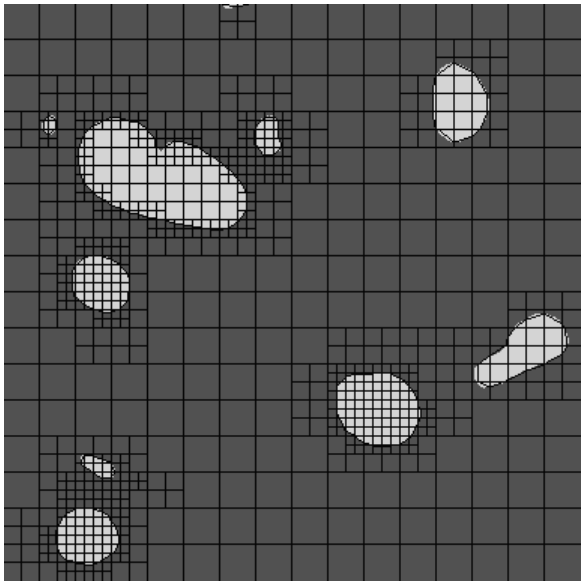


Figure 2.36: SnBat: After all IGFEM constraints were applied to mesh.

We report the number of elements after each IGFEM constraint was applied and calculate the cumulative factor (CF) of increase in the overall mesh size. For Figure 2.10a, the *interface* constraint more than doubled the mesh size. The other constraints have a moderate impact, with the *3-neighbor rule* constraint increasing the mesh size the least. After all constraints are applied, the final mesh is almost five times larger. For Figure 2.10b the *interface* constraint almost triples the mesh size, but the major impact is that of the *steep-gradient* constraint, which creates a mesh 25 times larger after all constraints are applied. We observe very similar behavior in Figure 2.10d, where the steep-gradient constraint leads to a mesh 20 times larger than the original one. In the case of Figure 2.10g, since we are dealing with a lower resolution image with smaller inclusions relative to overall dimensions, the impact of the constraints is less obvious: after all constraints are applied, the final mesh is three times the size of the original one.

2.7 Localization and Approximation of Material Interfaces

In the thermal analysis of a composite material, it is imperative that we determine not only where the interface is, but also a continuous function that can accurately approximate it. This information is needed in order to compute an accurate solution, and through segmentation we can achieve this.

Segmentation is a process that decomposes a tomographic image into its structural components by identifying the sets of voxels that constitute them. The most common approach (and simplest) is manual segmentation, where the user identifies the structures and categorizes them into domain features. There is also a plethora of automatic and semi-automatic methods, such as histogram-based methods, edge-detection, clustering, intelligent scissors, fast marching methods, level set methods, and many others [2]. None has become dominant over the others, as each has its own drawbacks. In general, segmentation can best be done on high-resolution and high-contrast images.

Like the classical algorithms, our approach to segmenting interfaces in heterogeneous elements is limited by the resolution of the image, the finite pixel size, and noise in the data, but must still be reliable and accurate. In our work we use two methods for accurately

capturing the curvature of the interface. One is by polynomial interpolation of points along the interface detected by shooting rays through the element, and the other is through a B-spline interpolation using control polygons and non-uniform intervals. The latter is known as non-uniform rational B-spline interpolation or NURBS. This approach also makes use of points detected along the interface through ray-shooting.

2.7.1 Interface Continuity

Continuity describes the degree of smoothness of a curve. Since the interface is approximated locally in each element, we can enforce only C^0 continuity across element boundaries. We do this in the following way. In each element we compute the intersection of element edges with the interface. We build a list of these intersection nodes as we go, and when we reach an adjacent element that already has a shared edge with an intersection node, we retrieve it from the list rather than recomputing it in the new element. This way we ensure a unique intersection node per edge, and rounding error does not create intersection nodes only one or two pixels apart on the same edge. Enforcing C^0 continuity this way is possible because Newton polynomial interpolation and NURBS interpolation always interpolate these end points that lie on the edges.

2.7.2 Polynomial Interpolation

For the polynomial interpolation we use Newton's method [31] so that the interpolating polynomial can be built incrementally as successive interpolation points are added. For a set of n points (x_i, y_i), the $(n-1)^{\text{th}}$ degree polynomial is

$$p_{n-1}(x) = y = a_1 + a_2(x - x_1) + a_3(x - x1)(x - x_2) + \cdots + a_n(x - x1)(x - x_2)\cdots(x - x_{n-1}),$$

where $a_1, a_2, a_3, \cdots, a_n$ are the polynomial coefficient to be determined.

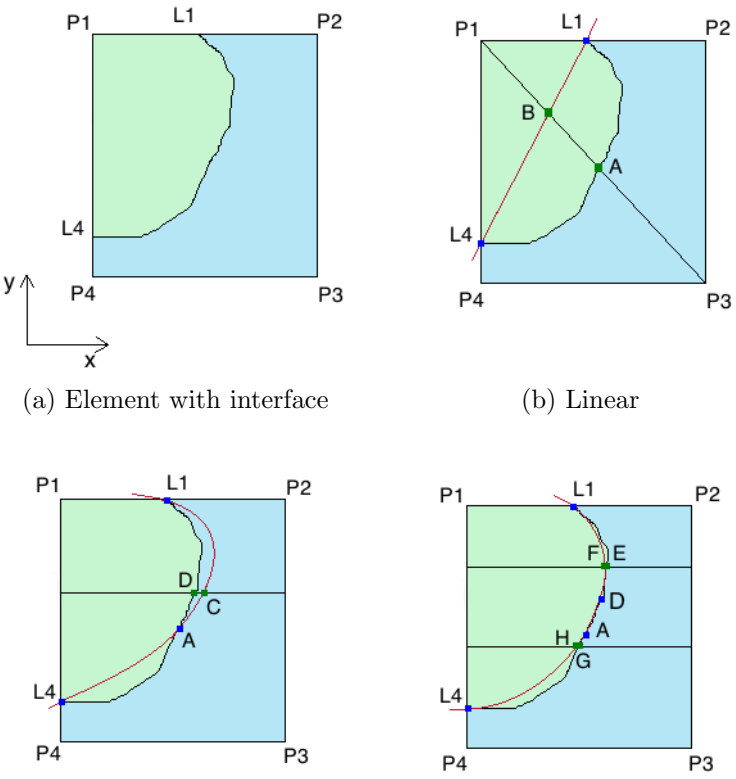
In determining the interface with polynomial interpolation, we choose the dependent variable of the function p_{n-1} based on whether the function has more variability in one direction or the other. In other words, if the support interval for x is larger than the support interval

for y, we use $y = p_{n-1}(x)$. If the support interval for y is larger than that for x, we use $x = p_{n-1}(y)$.

To approximate the interface with a linear polynomial, we need only the points of intersection of the element edges with the interface. After h-refinement is completed, there will always be at most two intersection points per element. To approximate the interface with a quadratic polynomial, we need one additional point that we determine by shooting a ray through the element and finding its intersection point with the interface. Where we shoot the ray through the element depends on which case of element-interface intersection we have (Figure 2.9). If we are in a corner or edge case, we shoot a ray along the diagonal of the element that will intersect the interface. If we are in the horizontal or vertical case the ray is shot along vertically or horizontally, respectively, at the midpoint of the edge. In the case of diagonal interfaces, we shoot a ray along the opposite diagonal. To approximate the interface with a cubic polynomial, we need a total of four points, of which two are interior. The interior points are determined in a similar way that we determined the one interior point for quadratric polynomials, except all the rays are shot at one-third and two-thirds horizontally or vertically, on the edge along which there is greater variability for the x or y variable (the edge corresponding to the larger of Δx and Δy).

We measure how well the interpolating polynomials approximate the interface by passing additional rays through the element and computing the intersection of these rays with the interface and with the polynomial approximation. For each additional ray shot through the element, we compute the distance between these two intersection points. If that distance exceeds a preset tolerance, then the we declare the approximation not good enough, triggering h or q refinement.

Details of the polynomial approximation procedure are best explained through the use of an example. Let us consider the element shown in Figure 2.37, for which we want the interface to be approximated by a higher-order polynomial, say a cubic. In this element, the north-west corner belongs to a different material than the rest of the domain (corner $P1$ is in one material, while corners $P2, P3, P4$ are in a different one). Through a search along each edge we determine the intersection between the element and the interface. In this example, we find that the edge $P1 - P2$ is intersected at location $L1$, and edge $P1 - P4$ at location



(a) Element with interface

(b) Linear

(c) Quadratic

(d) Cubic

Figure 2.37: Determining polynomial approximation of interface.

$L4$. We also determine based on the lengths of $P1 - L1$ and $P1 - L4$ whether we should express our polynomial as a function of the x coordinate or the y coordinate.

With this information we proceed with the approximation of the interface with a linear polynomial. First, from the corner $P1$ to the diagonally opposed one, we do a log-search to determine the intersection of the interface with this diagonal. We denote this intersection point by A. Then, we search for the intersection of a straight line passing through $L1$ and $L4$, with the diagonal $P1-P3$. This intersection point we call B. If the straight line $L1-L2$ closely matches the interface, then the distance between point A and point B should be very small (we choose the treshold 1 pixel). In this example, a linear polynomial would be a gross approximation of the interface and we proceed with trying to approximate it with a quadratic. We build the quadratic by Newton interpolation through the already determined points A, $L1$ and $L4$. Along the longer dimension, which in this case is y because the distance

between $P1$ and $L4$ is greater than the distance between $P1$ and $L1$, we find the midpoint between the corner $P1$ and the interface node $L4$. We construct a straight line parallel to edges $P1 - P2$ and $P4 - P3$. Using this line we first determine its intersection with the quadratic passing through $L1, A, L4$ and we denote this point by C. Along the same line, we do a log-search to determine its intersection with the interface, point D. We then compute the distance between C and D and check whether it meets our pre-established tolerance of 1. It does not, so we will try next an approximation of the interface with a cubic polynomial. Again, along the longer dimension we construct two parallel lines to the top and bottom edges of the element, lines that pass through points at $\frac{1}{3}$ and $\frac{2}{3}$ between nodes $P1$ and $L4$. We determine their intersections with the interface and find them to be E and G. Then we look for their intersection with a cubic polynomial passing through the points $L4, A, D, L1$, and we denote them by F and H. If the distances between E and F, and G and H are within the treshold, we can conclude the curvature can be sufficiently well approximated by a cubic polynomial, and the process ends for this element.

2.7.3 Results

We will now summarize the results we obtained by running the meshing algorithm for a series of images at various resolutions when allowing various choices for approximating the interface with polynomials. We chose four images that have geometry representative of the materials we are considering: Figures 2.10a, 2.10b, 2.10d and 2.10e. For each of these images we show statistical information in Tables 2.4, 2.6, 2.7, and 2.5, respectively. This information contains the total number of elements in the mesh, the number of heterogeneous elements, what kind of polynomial approximation was used to match the interface in those elements, and the percentage by which the mesh size was reduced when allowing up to quadratic and cubic polynomial approximations. We also plot this data in histograms in Figures 2.38, 2.40, 2.41 and 2.39. The different clusters in the histograms represent the different resolutions preset for running the meshing algorithm. Sometimes the minimum size element is reached, sometimes not. When using high-order polynomial approximations such as cubics, q-refinement sometimes suffices and h-refinement down to minimum h is not

triggered. The height of each bar in the histogram represents the total number of elements for a given resolution, when a given q-refinement was allowed. Each of these bars also contains information about how many of the elements were homogeneous (drawn in light green color), and how many were heterogeneous and of what type: light blue for linear q-refinement, orange for quadratic q-refinement, and dark blue for cubic q-refinement.

Resolution	Linears	Quadratics		Cubics		
	# of elems	# of elems	mesh reduction form linears	# of elems	mesh reduction from linears	mesh reduction from quadratics
MAX = 250 MIN = 63	46 29 L	43 5 L + 23 Q	7 %	25 3 L + 5 Q + 8 C	46 %	42 %
MAX = 125 MIN = 63	136 51 L	85 5 L + 30 Q	38 %	73 5 L + 22 Q + 6 C	46 %	14 %
MAX = 125 MIN = 31	274 109 L	103 6 L + 33 Q	62 %	73 5 L + 22 Q + 6 C	73 %	30 %
MAX = 63 MIN = 16	430 125 L	307 13 L + 56 Q	29 %	262 7 L + 33 Q + 13 C	39 %	15 %

Table 2.4: Circles: 1000 × 1000 px^2. Number of elements in mesh for various h- and q-refinements.

Resolution	Linears	Quadratics		Cubics		
	# of elems	# of elems	mesh reduction form linears	# of elems	mesh reduction from linears	mesh reduction from quadratics
MAX = 64 MIN = 16	216 130 L	165 51 L + 55 Q	24 %	148 43 L + 40 Q + 15 C	31 %	10 %
MAX = 64 MIN = 8	404 213 L	198 63 L + 54 Q	51 %	157 46 L + 42 Q + 14 C	61 %	21 %
MAX = 32 MIN = 16	294 135 L	294 68 L + 67 Q	0 %	294 67 L + 53 Q + 15 C	0 %	0 %
MAX = 32 MIN = 8	471 212 L	352 93 L + 64 Q	25 %	327 81 L + 55 Q + 11 C	31 %	7 %

Table 2.5: Fibers: 256 × 256 px^2. Number of elements in mesh for various h- and q-refinements.

The choice of q-refinement affects the mesh in different ways depending on the geometry of the problem. For example, for Figure 2.10a changing the resolution of the image can change the number of elements in the mesh quite significantly: we see percentages such as 38% and 62% when using quadratics over linears, and 46% and 73% when using cubics over linears. We also see a reduction in the number of mesh elements by going from quadratics to cubics. Some of these gains can be as high as 42% or 30%, or closer to 14-15%. Similar

Resolution	Linears	Quadratics		Cubics		
	# of elems	# of elems	mesh reduction form linears	# of elems	mesh reduction from linears	mesh reduction from quadratics
MAX = 250 MIN = 63	184 131 L	142 70 L + 38 Q	23 %	142 70 L + 36 Q + 2 C	23 %	0 %
MAX = 125 MIN = 31	304 194 L	163 75 L + 38 Q	46 %	157 70 L + 37 Q + 2 C	48 %	4 %
MAX = 63 MIN = 16	442 220 L	298 96 L + 42 Q	33 %	292 91 L + 41 Q + 2 C	34 %	2 %
MAX = 63 MIN = 8	527 269 L	301 98 L + 42 Q	43 %	292 91 L + 41 Q + 2 C	46 %	3 %

Table 2.6: Channels: 1000×1000 px^2. Number of elements in mesh for various h- and q-refinements.

Resolution	Linears	Quadratics		Cubics		
	# of elems	# of elems	mesh reduction form linears	# of elems	mesh reduction from linears	mesh reduction from quadratics
MAX = 350 MIN = 87	76 52 L	76 25 L + 27 Q	0 %	76 25 L + 27 Q + 0 C	0 %	0 %
MAX = 175 MIN = 21	178 77 L	106 25 L + 27 Q	40 %	106 25 L + 27 Q + 0 C	40 %	0 %
MAX = 175 MIN = 8	310 109 L	106 25 L + 27 Q	66 %	106 25 L + 27 Q + 0 C	66 %	0 %
MAX = 63 MIN = 16	484 137 L	283 25 L + 35 Q	42 %	283 25 L + 35 Q + 0 C	42 %	0 %

Table 2.7: Microvascular: 700×700 px^2. Number of elements in mesh for various h- and q-refinements.

improvements can also be seen with the composite material depicted in Figure 2.10e, which also contains circular inclusions. Reductions in the mesh size can be as high as 51% and 61% when going from linears to quadratics, and linears to cubics, respectively. The reductions for going from quadratics to cubics are less pronounced, with percentages of 21% and 7%. There is also a 0% gain going between the different q-refinements for a particular mesh resolution: MAX = 32, and MIN = 16. The total number of elements in the mesh does not change, 294; the only change is the type of heterogeneous elements. Out of the 294 elements, 135 are heterogeneous. In the case of linear q-refinement, all of them are obviously approximated with linear polynomials. When we allow up to quadratic q-refinement, 67 of these elements have the interface better approximated with quadratics. Allowing up to cubic q-refinement, 15 trigger cubic polynomial approximation, and 53 quadratic.

The geometry of the problem also influences whether a certain type of q-refinement is used and how often. For sine-like inclusions, we considered a manufactured image, Figure 2.10b,

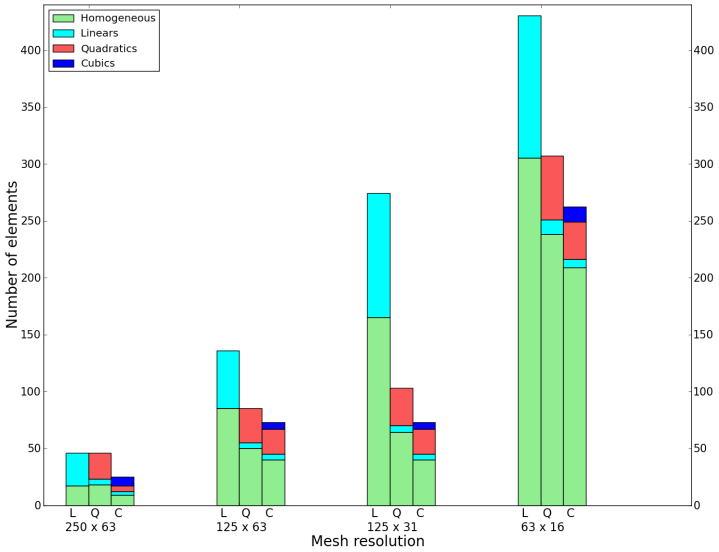


Figure 2.38: Comparing number of elements and types of elements for Circles at various resolutions. Total number of elements is given by homogeneous elements and heterogeneous with linear polynomial approximations (L), up to quadratic polynomial approximations (Q), and up to cubic polynomial approximations (C) of interface.

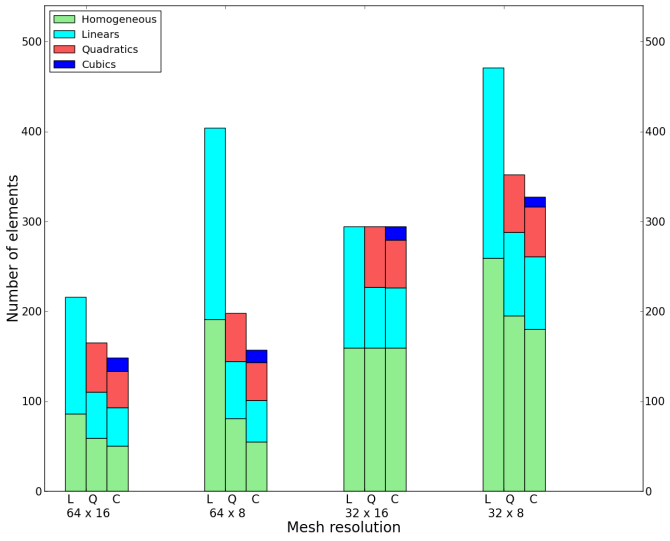


Figure 2.39: Comparing number of elements and types of elements for Fibers at various resolutions. Total number of elements is given by homogeneous elements and heterogeneous with linear polynomial approximations (L), up to quadratic polynomial approximations (Q), and up to cubic polynomial approximations (C) of interface.

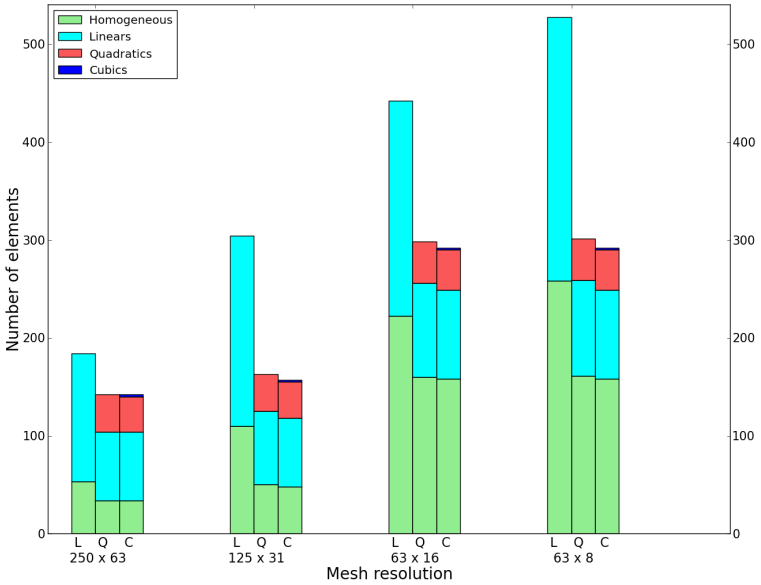


Figure 2.40: Comparing number of elements and types of elements for Channels at various resolutions. Total number of elements is given by homogeneous elements and heterogeneous with linear polynomial approximations (L), up to quadratic polynomial approximations (Q), and up to cubic polynomial approximations (C) of interface.

and a tomographic one (containing noise), Figure 2.10d. There is still a significant reduction in the mesh size by going from linears to quadratics or cubics. For the artificial image we see reductions as high as 46% for quadratic q-refinement and 48% for cubic q-refinement. For the tomographic image, the reductions are as high as 66% for both quadratics and cubics. However, the reductions going from quadratics to cubics are either very small, 3% or 4% for the artificial image, or non-existent, 0% for the tomographic one. We attribute these small percentages to the curvature of the interface. The shape of the interface is sufficiently complicated that linears do a poor job without additional h-refinement, but simple enough for quadratics to do a very good job. For both images, we see that the number of heterogeneous elements hardly changes when different h-resolutions are employed. The histograms also illustrate well the lack of significant change from quadratics to cubics. Figure 2.40 shows barely any blue bars, while Figure 2.41 shows none at all.

The curvature of the interface and the degree of the polynomial are closely inter-related and dictate the level of refinement. For all the sets of images used we recognize that hq-

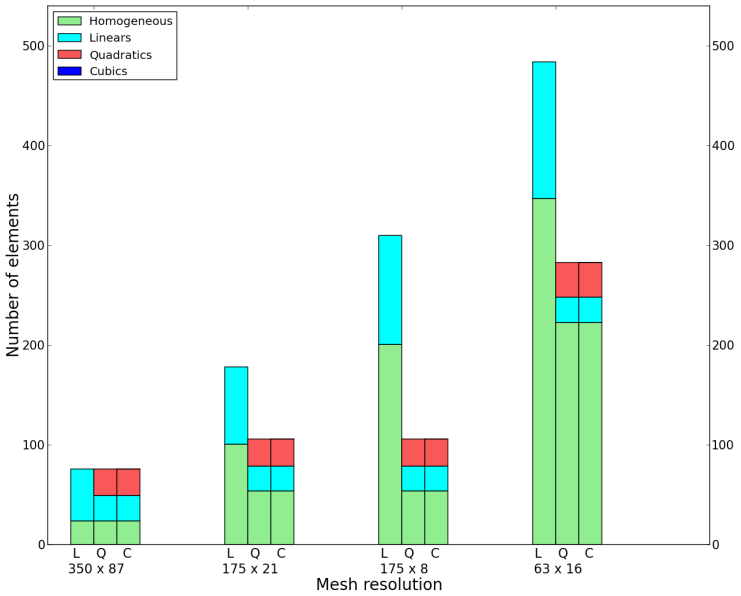


Figure 2.41: Comparing number of elements and types of elements for Microvascular at various resolutions. Total number of elements is given by homogeneous elements and heterogeneous with linear polynomial approximations (L), up to quadratic polynomial approximations (Q), and up to cubic polynomial approximations (C) of interface.

refinement creates a trade-off between the mesh resolution and the accuracy with which we want to approximate the interface. One can use low-order approximations to the interface, but would have to do so at the cost of a more refined mesh. On the other hand, one could spend more effort on obtaining an accurate representation of the interface, and then use a coarser mesh. Choosing one over the other can thus have both advantages and disadvantages.

2.7.4 NURBS or Non-Uniform Rational B-Splines Interpolation

Another way to represent the material interfaces is through the use of B-splines, in particular non-uniform rational B-splines, or NURBS. They have become the standard in representing complicated geometry in computer graphics mainly due to the fact that they are very efficient for data smoothing, local modification, and effective data storage [26, 32, 34]. B-splines can be used either to interpolate or to approximate interface curves and surfaces; however, in the context of this work, we consider only B-spline interpolation.

NURBS are defined by

- order of the curve,
- weighted control points,
- knot vector.

The following relationships exist among these three NURBS characteristics:

$$\text{number of knots} \;=\; \text{number of control points} \;+\; \text{order of curve},$$

$$\text{number of control points} \;\geq\; \text{order of curve},$$

$$\text{order of curve} \;\geq 2.$$

Order of the curve or the spline degree refers to the degree of the piecewise polynomials used. The choice of the degree, which we call k, directly affects how well NURBS fit the interface. For simplicity of implementation while still retaining reasonably powerful fitting ability, we use quadratic NURBS, which have order 3 (cubics would have order 4, and linears order 2).

Control points are similar to nodal values in interpolation. The points are not interpolated, but rather approximated, unless the knot values have multiplicity $k-1$. Thus, to force the curve to go through the endpoints, the end points in the knot vector must be repeated k times. The number of control points determines the number of degrees of freedom available in the spline fitting. Weights control the influence of each control point on the resulting curve.

The knot vector is a parametric sequence of values associated with the control points and determines where and how these affect the NURBS curve. The knot vector directly affects the B-spline basis functions, which in turn determine the shape of the curve. The knot vector values must be in ascending order, and their magnitudes do not matter.

Given a knot vector $T = \{t_0, t_1, \ldots, t_i, t_{i+1}, \ldots, t_{m+p+1}\}$, a set of weights $\{w_i\}$, a set of control points $\{P_i\}$, and normalized B-spline basis functions $\{N_{i,p}(t)\}$ of degree q, parametric equation for the NURBS of degree q is

$$\gamma(t) = \frac{\sum_{i=0}^{m} w_i P_i N_{i,p}(t)}{\sum_{i=0}^{m} w_i N_{i,p}(t)},$$

where the B-spline basis functions are defined recursively by

$$N_{i,p}(u) = \frac{t - t_i}{t_{i+p} - t_i} N_{i,p-1}(t) + \frac{t_{i+p+1} - t}{t_{i+p+1} - t_{i+1}} N_{i+1,p-1}(t),$$

and

$$N_{i,0}(t) = \begin{cases} 1 & \text{if } t_i \leq t \leq t_{i+1}, \\ 0 & \text{otherwise.} \end{cases}$$

In our analysis we consider only quadratic NURBS with the following knot sequence:

$$t = [0, 0, 0, \frac{1}{2}, 1, 1, 1].$$

They are determined by the following recursive relation for the basis functions:

$$\begin{aligned} N_{i,2}(t) &= \frac{t - t_{i-1}}{t_{i+1} - t_{i-1}} \frac{t - t_{i-1}}{t_i - t_{i-1}} N_{i,0}(t) \\ &+ \Big(\frac{t - t_{i-1}}{t_{i+1} - t_{i-1}} \frac{t_{i+1} - t}{t_{i+1} - t_i} + \frac{t_{i+2} - t}{t_{i+2} - t_i} \frac{t - t_i}{t_{i+1} - t_i} \Big) N_{i+1,0}(t) \\ &+ \frac{t_{i+2} - t}{t_{i+2} - t_i} \frac{t_{i+2} - t}{t_{i+2} - t_{i+1}} N_{i+2,0}(t). \end{aligned}$$

To determine the control points P_i, we solve a system of $m + 1$ linear equations of the form

$$\sum_{i=0}^{m} P_i R_{i,p}(\hat{t_k}) = Q_k, \quad k = 0, \cdots, m,$$

where the rational functions $R_{i,p}(t)$ are defined by

$$R_{i,p}(t) = \frac{w_i N_{i,p}(t)}{\sum_{i=0}^{m} w_i N_{i,p}(t)},$$

and the right-hand side vector Q_k is a vector of sampled points that the NURBS curve is to interpolate. In our mesh, we use the same methodology in determining the control points as

in determining points on the interface for polynomial approximation. We shoot rays through the element and find the intersection of each ray with the interface. In the case of quadratic NURBS, four points suffice: two are the end points, where the interface intersects the edges of the element, and two are interior points determined in the same way as in the case of cubic polynomials.

The parametric values $\hat{t}_k$ are determined using the chord length parameterization method, with

$$\begin{aligned} \hat{t}_0 &= 0, \\ \hat{t}_m &= 1, \\ \hat{t}_i &= \hat{t}_{i-1} + \frac{|Q_i - Q_{i-1}|}{\sum_{j=0}^{m} |Q_j - Q_{j-1}|}, \quad i = 1, \cdots, m-1. \end{aligned}$$

There are several other parameterization methods, such as the uniform method, the centripetal method, and the universal method. Each of these methods has its own advantages and disadvantages. We chose the chorld length method because its disadvantage, which is that it becomes wiggly for long chords, does not apply in the case of our short chords. The uniform method works only with uniformly spaced control points, and we cannot guarantee that. The universal method works well with long chords, but not with short chords for which peaks and loops occur. The centripetal method is an extension of the chord length method that tends to even out distances between two parameters, thus sharing the same disadvantages with the uniform and universal methods.

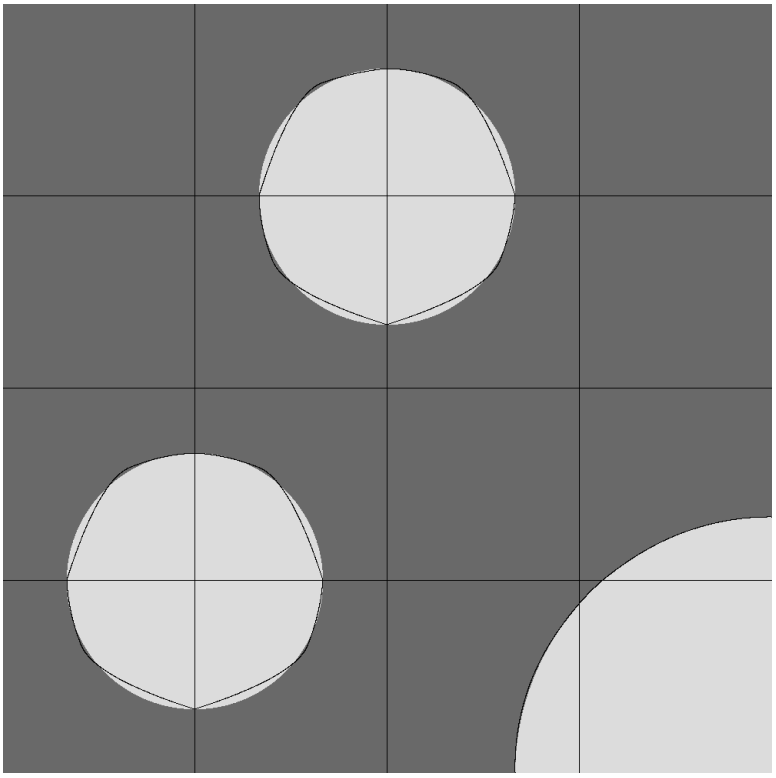
How well does a NURBS curve approximate the interface?

To measure how well a NURBS curve approximates the interface, we employ the same methodology as in the case of the polynomial approximations. We first determine the control points for the NURBS curve from sampling points and then build the NURBS curve. The curve is computed as $y = f(x)$ or $x = g(y)$, based on whether the support interval Δx or Δy is longer. Along the longer support interval we then shoot a ray at half distance and determine the intersection of this ray with the NURBS curve and with the interface.

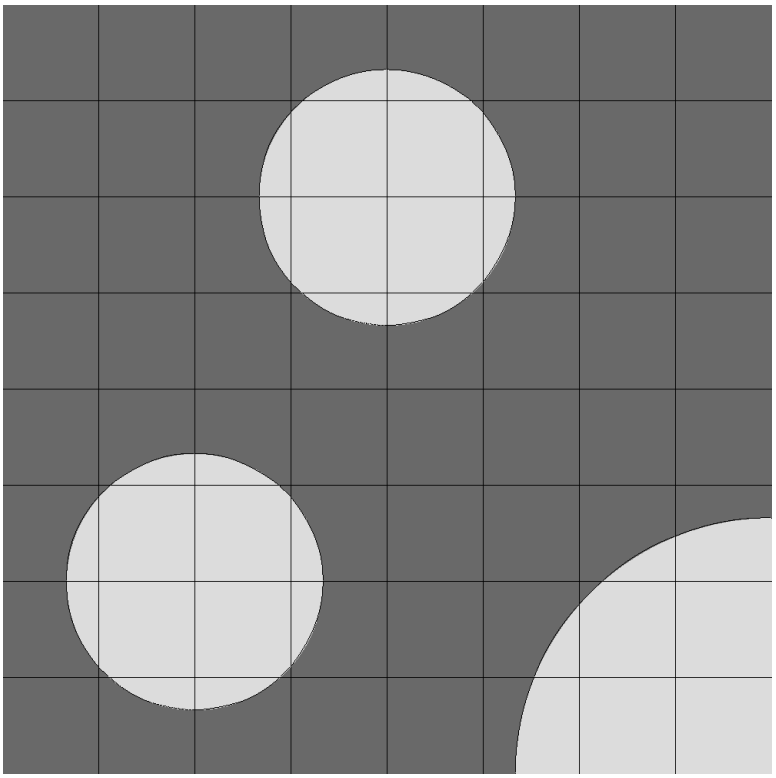
The difference between the two intersection points is compared to a tolerance to determine whether the B-spline interpolation is sufficiently accurate.

Figure 2.42a shows a rough approximation of interface with a non-uniform rational B-spline. No h-refinement was allowed in this case for the NURBS. By permitting refinement of the mesh, the NURBS can fit the curvature of the circles better, as can be seen from Figure 2.42b, for which the allowed tolerance is 3 pixels.

Table 2.8 shows for various images how the choice for approximating the interface affects the number of elements in the mesh. We allow only hq-refinements and no other constraints, since we want to capture accurately the effect of B-splines versus polynomials. For a fixed minimum and maximum element size, we first fit the interface with NURBS. Afterwards, we try polynomials of degree one, then up to degree two, then up to degree three. The first three images are artificially created. The remaining four are slices of 3D tomographic datasets microvascular, fibers, and battery composites (silicon and tin). As expected, linear polynomials always lead to the largest meshes. NURBS are always better than linears, almost always as good as quadratics if not better, and sometimes even as good as cubics. Whether NURBS are better than polynomials depends on the geometry of the inclusions and their size relative to the overall image size. Figure 2.44 plots the data from this table. For each image, we show the number of elements in the mesh. The height of each bar represents how many elements were created for that particular choice of approximating the interface: NURBS, linear polynomials, up to quadratic polynomials, or up to cubic polynomials.

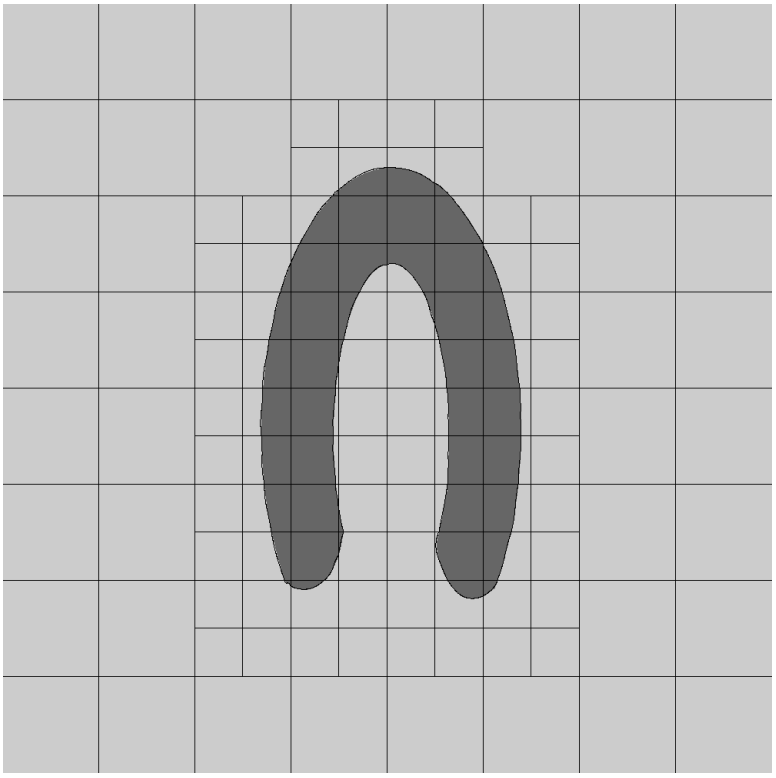


(a)

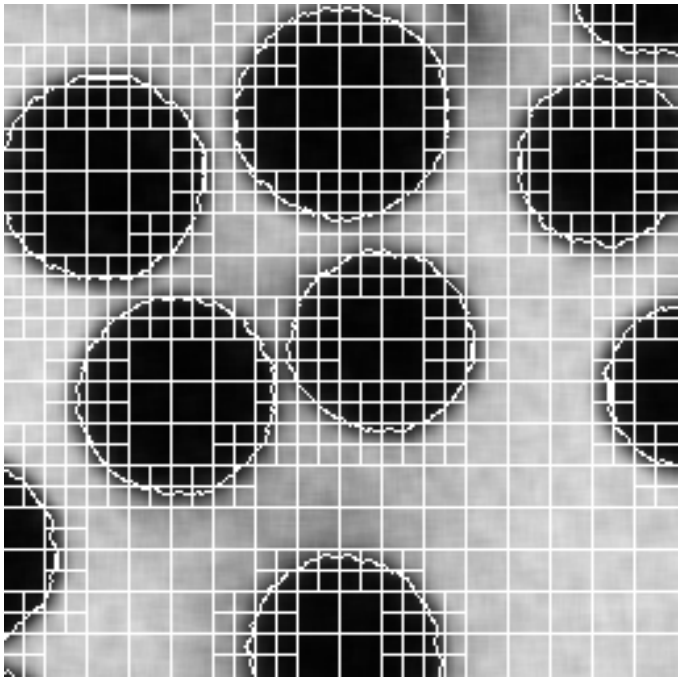


(b)

Figure 2.42: NURBS are used to approximate interface. (a) For coarse mesh, NURBS curve does not approximate interface well. (b) Allowing for further h-refinement, NURBS achieve better fit.



(a)



(b)

Figure 2.43: NURBS are used to approximate the interface. (a) horseshoe-like geometry with coarse mesh (b) slice of a real dataset.

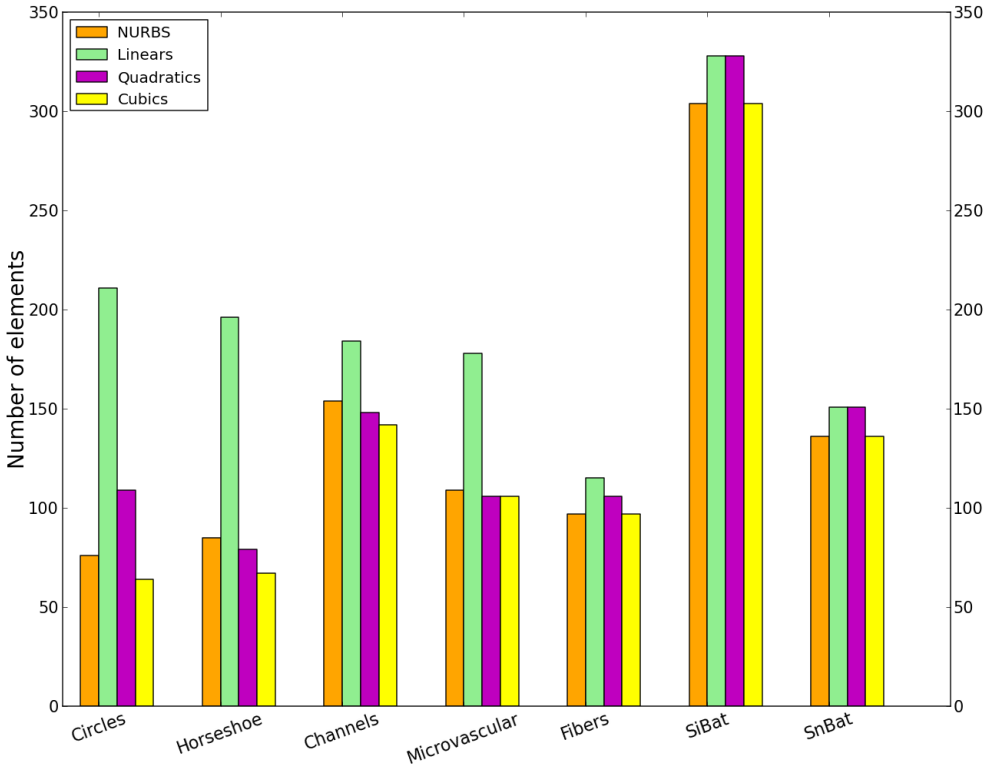


Figure 2.44: Number of elements in mesh for different interface approximations (NURBS or polynomials). Only *hq*-refinement is applied to mesh, and no other constraints.

Images	Dimension in px^2	Resolution	NURBS - # of elems	Linears - # of elems	Quadratics - # of elems	Cubics - # of elems
Circles	1000×1000	MAX = 250 MIN = 25	76	211	109	64
Horseshoe	1000×1000	MAX = 250 MIN = 25	85	196	79	67
Channels	1000×1000	MAX = 250 MIN = 25	154	184	148	142
Microvascular	700×700	MAX = 175 MIN = 21	109	178	106	106
Fibers	256×256	MAX = 128 MIN = 8	97	115	106	97
SiBat	512×512	MAX = 64 MIN = 16	304	328	328	304
SnBat	384×384	MAX = 192 MIN = 6	136	151	151	136

Table 2.8: Comparison of number of mesh elements for NURBS and polynomial interpolation for various images.

CHAPTER 3

Interface-based Generalized Finite Element Analysis with Quadrilateral Meshes

The focus of this chapter is on solving the heat-conduction equation numerically on a domain with inclusions, such as active-cooling microvascular channels inside, as shown schematically in Figure 3.1. To solve the variational or weak form of the partial differential equation (PDE), we employ the finite element method with a non-conforming mesh. In particular, we use the standard Galerkin method to obtain the weak form of the PDE using multiplication by test functions and integration by parts.

Some of the early work on finite element analysis using non-conforming meshes has been done in the context of crack growth [40, 13], where growing cracks are modeled without the need for remeshing. Later work [7] has been done in the context of heterogeneous materials. For these materials, thermal and mechanical analysis can be done by adding discontinuous enrichment functions to the finite element approximation in elements traversed by a crack or interface. This process is known as *enrichment* of the solution within the element. In this chapter we focus on heterogeneous materials with interfaces and the resulting finite element methodology, known as the Interface-based Generalized Finite Element Method (IGFEM) [50, 49].

The remainder of this chapter is organized as follows. Section 3.1 introduces the thermal problem (PDE) and the standard Galerkin methodology to solve it. In Section 3.2 we briefly review the generalized finite element method (GFEM) and introduce the concept of solution space enrichment. Section 3.3 describes IGFEM in the context of quadrilateral meshes defined by images, discusses the use of isoparametric elements in integration, and the domains of integration. Section 3.4 shows results for the thermal problem as applied to various geometries inferred from images.

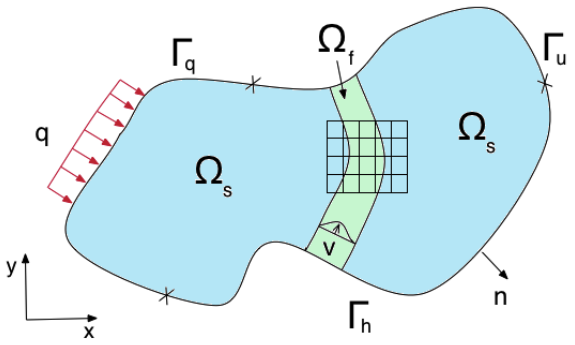


Figure 3.1: Schematic configuration of 2D domain geometry with boundary conditions. Also illustrated is a portion of a mesh that does not conform to the phase interfaces.

3.1 Problem Statement

As shown schematically in Figure 3.1, the open domain $\Omega = \Omega_s \cup \Omega_f \subset \mathbb{R}^2$ is composed of two mutually exclusive domains representing the solid phase (Ω_s) and the fluid phase (Ω_f): $\Omega_s \cap \Omega_f = \emptyset$. Its closure is denoted by $\bar{\Omega}$. The boundary $\Gamma = \bar{\Omega} - \Omega$ is divided into three partitions: Γ_u for Dirichlet boundary conditions with prescribed temperature $\bar{u} : \Gamma_u \to \mathbb{R}$, Γ_q for Neumann boundary conditions with heat flux $q : \Gamma_q \to \mathbb{R}$, and Γ_h for Robin boundary conditions with heat transfer coefficient h, and ambient temperature $u_\infty : \Gamma_h \to \mathbb{R}$. $\Gamma = \overline{\Gamma_u \cup \Gamma_s \cup \Gamma_h}$ and $\Gamma_u \cap \Gamma_q \cap \Gamma_h = \emptyset$. $\mathbf{n}$ is the outward unit normal of the boundary.

For a thermal conductivity $\kappa : \bar{\Omega} \to \mathbb{R}$, fluid density $\rho : \Omega_f \to \mathbb{R}$, fluid specific heat $c_p : \Omega \to \mathbb{R}$, velocity field $\mathbf{v} : \Omega_f \to \mathbb{R}$ and a heat source $f : \Omega \to \mathbb{R}$, the strong form of the problem is expressed as

- Find $u : \bar{\Omega} \to \mathbb{R}$ such that

$$
\begin{aligned}
-\nabla \cdot (\kappa \nabla u) + \rho c_p \mathbf{v} \cdot \nabla u &= f \text{ on } \Omega, \\
u &= \bar{u} \text{ on } \Gamma_u, \\
\kappa \nabla u \cdot \mathbf{n} &= q \text{ on } \Gamma_q, \\
\kappa \nabla u \cdot \mathbf{n} &= h(u_\infty - u) \text{ on } \Gamma_h.
\end{aligned}
$$

If $\mathbf{v} = 0$ or $\Omega = \Omega_s$, the problem reduces to the Poisson equation.

Given the function spaces $\mathcal{U} = \{u : u|_{\Gamma_u} = \bar{u}\} \subset H^1(\bar{\Omega})$, and $\mathcal{W} = \{w : w|_{\Gamma_u} = 0\} \subset H^1(\bar{\Omega})$, the problem can be described in its weak form as

- Find $u \in \mathcal{U}$ such that

$$a(u, w) + a(u, w)_{\Gamma_h} = (w, f) + (w, q)_{\Gamma_q} + (w, u_\infty)_{\Gamma_h}, \forall w \in \mathcal{W}, \tag{3.1}$$

where the linear and bilinear forms are given by

$$\begin{aligned}
a(u, w) &= \int_\Omega \nabla w \cdot (\kappa \nabla u)\, d\Omega + \int_{\Omega_f} w \rho c_p \mathbf{v} \cdot \nabla u\, d\Omega, \\
a(u, w)_{\Gamma_h} &= \int_{\Gamma_h} hwu\, d\Gamma_h, \\
(w, f) &= \int_\Omega wf\, d\Omega, \\
(w, q)_{\Gamma_q} &= \int_{\Gamma_q} wq\, d\Gamma_q, \\
(w, u_\infty)_{\Gamma_h} &= \int_{\Gamma_h} hwu_\infty\, d\Gamma_h.
\end{aligned}$$

The Galerkin formulation of Equation (3.1) can be written as

$$a(u^h, w^h) + a(u^h, w^h)_{\Gamma_h} = (w^h, f) + (w^h, q)_{\Gamma_q} + (w^h, u_\infty)_{\Gamma_h}, \forall w^h \in \mathcal{W}^h, \tag{3.2}$$

where subspaces $\mathcal{U}^h \subset \mathcal{U}$ and $\mathcal{W}^h \subset \mathcal{W}$ are defined as

$$\mathcal{W}^h = \{w^h : w^h|_{\Gamma_u} = 0\},$$

$$\mathcal{U}^h = \{u^h : u^h = w^h + t^h, t^h|_{\Gamma_u} = \bar{u}, w^h \in \mathcal{W}^h\},$$

and h measures maximum element size.

For a set of n Lagrangian shape functions $N_i(x)$ and for a discretized domain Ω into m finite elements, Equation (3.2) can be used in the standard finite element method approximation in each element:

$$u^h(x, y) = \sum_{i=1}^{n} N_i(x, y) u_i.$$

Additional terms will be introduced into this approximation, as we will describe enhance-

ments to the finite element method for composite materials meshed with structured and non-conforming meshes.

3.2 Partition of Unity and Generalized/Extended Finite Element Methods

Finite elements can be enriched by requiring the sum of the shape functions to be unity. This concept is known as a *partition of unity* [9, 24].

A partition of unity in a domain Ω is a set of functions $\phi_I(x, y)$ such that

$$\sum_{\forall I} \phi_I(x, y) = 1, \forall x, y \in \Omega.$$

Any function $\Phi(x, y)$ can be reproduced by a product of the partition of unity functions with the function itself. The solution approximation is enriched by adding the enrichment part u^{enr} to the finite element approximation u^{FE},

$$u^h(x) = \underbrace{\sum_{\forall I} N_I(x, y) u_I}_{u^{FE}} + \underbrace{\sum_{\forall I} \phi_I(x, y) \Phi(x, y) q_I}_{u^{enr}},$$

where $N_I(x, y)$ and u_I are the standard basis functions, and standard nodal values, respectively, and q_I are unknown nodal values that adjust the enrichment such that it best approximates the solution. $\Phi(x, y)$ is an enrichment function based on some prior knowledge about the solution. The standard and enrichment basis functions $N_I(x, y)$ and $\phi_I(x, y)$ do not have to be the same, but generally they are chosen to be so. A typical choice for the enrichment basis functions are the Lagrangian finite element shape functions. They satisfy the partition of unity property, which is essential for convergence.

Melenk and Babuska [9] developed a method based on this concept, called the Partition of Unity Method or PUM. The motivation was the need for new techniques that could solve problems where classical FEM failed or was prohibitively expensive. This work is usually recognized as the precursor of the Generalized Finite Element Method (GFEM) [10] and

Extended Finite Element Method (XFEM) [40]. GFEM and XFEM, which are essentially the same, but named differently by the two different research groups that developed them, simplify the modeling of discontinuous phenomena by allowing construction of elements that do not conform to the problem geometry. This allows the finite element mesh to be completely independent of the material morphology. GFEM/XFEM has been used on a wide range of problems, such as crack propagation in fracture mechanics [40, 13, 21], structural problems [23, 48], phase interface/change problems [7, 39, 53], and multiscale methods [27].

A-priori knowledge about the character of the solution (e.g., singularity, discontinuity, boundary layer) can be incorporated into the classical approximation to improve its accuracy and to recover optimal convergence that may otherwise be lost by the use of a non-conforming mesh. This knowledge is introduced by means of enrichment functions, which can range from simple polynomials to sophisticated functions such as Westergaard functions. The enrichment functions are added at the nodes of the original mesh.

For elements with interfaces, XFEM and GFEM have been used together with level set methods that can determine the location of the interface. In XFEM, the enrichment part of the approximation is given by

$$u^{enr}(x,y) = \sum_I N_I^\star(x,y)|f(x,y)|q_I,$$

where q_I is the coefficient at the element nodes that must be determined, $N_I^\star(x,y)$ is a partition-of-unity function, and $f(x,y)$ is a signed distance function, which gives the shortest distance to the discontinuity. The simplest enrichment function is a tent function, which satisfies the partion of unity requirement. The absolute value of $f(x)$ can be interpreted as the absolute value of the level set function. The zero-level of the level-set function is the discontinuity. The idea of using a ridge to model discontinous gradients was first introduced by Kongrauz and Belytschko [33].

In the context of the problem presented in Section 3.1, the Galerkin method cannot capture the gradient discontinuity at the interface, thus introducing substantial error and loss of the optimal convergence rate. These issues can be resolved by adding local enrichment functions ϕ_{ij} at the nodes of the original mesh for the elements intersected by an interface,

$$u^h(x,y) = \sum_{i=1}^{n} N_i(x,y)\bar{u} + \sum_{i=1}^{n} N_i(x,y) \sum_{j=1}^{n_{enr}} \phi_{ij}(x,y)\hat{u}_{ij}, \tag{3.3}$$

where $\{\phi_{ij}(x,y) : x,y \to \mathbb{R} | N_i(x,y) \neq 0\}$.

There are several issues with GFEM. In some cases, the enrichment must be applied also at the nodes of adjacent elements (also known as blending elements), since the partition of unity may be unable to provide a smooth transition between standard FEM elements and those intersected by the interface. Another issue with GFEM involves the application of Dirichlet boundary conditions at the enriched nodes of the mesh. Such boundary conditions cannot be directly enforced except through the use of techniques such as the penalty method or Lagrange multipliers. Computing the quadrature in the enriched elements can also be problematic. Using the same order of Gauss points as in the standard FE approximation leads to a suboptimal rate of convergence, and using higher-order Gauss quadrature does not improve the accuracy [52]. The most commonly adopted approach is to split the domain of integration into subdomains that adhere to the phase interface, where the discontinuities lie.

To overcome some of these issues, while continuing to maintain a structured and non-conforming mesh, the GFEM method has been recently adapted into a new methodology that applies enrichment functions at additional nodes defined by the intersection of the material interface and the nonconforming element [50, 49]. This approach is called the Interface-based Generalized Finite Element Method or IGFEM, and is the cornerstone of this research.

3.3 IGFEM for Quadrilateral Meshes

Thus far IGFEM has been used only with triangular meshes in 2D and tetrahedral meshes in 3D, with analytically specified interfaces and inclusions. IGFEM has not been used with quadrilateral meshes (2D) or hexahedral meshes (3D), or in the context of tomographic data, where the geometry of the materials and inclusions is determined based on this image representation.

The main difference between GFEM and IGFEM is how the enrichment functions are applied. In GFEM, they are applied at the nodes of the original mesh, whereas in IGFEM they are applied at the nodes of the intersections between the edges of an element and an interface. In the context of working with tomographic data, however, developing a 2D quad-based (hex-based in 3D) IGFEM to take advantage of the pixel/voxel representation of the micro-CT image is a more natural approach. Quad and hex-meshes not only have higher order accuracy, but are also applicable irrespective of the heterogeneity of the material. The perceived greater complexity of determining intersections of material interfaces with hexahedral elements is more manageable in the context of pixel- or voxel- aligned meshes and pixel- or voxel- defined interfaces, since the finite element mesh can be a direct subset of the underlying pixel or voxel mesh.

The GFEM formulation from Equation 3.3 changes for IGFEM to

$$u^h(x,y) = \sum_{i=1}^{n} N_i(x,y)u_i + \sum_{i=1}^{n_{en}} s_j\psi_j(x,y)\alpha_j, \tag{3.4}$$

where (x, y) is a given spatial position in the problem domain, u^h is the approximate thermal or structural solution, n and n_{en} are the numbers of basis functions and enrichment functions, respectively, $N_i(x, y)$ and $\psi_j(x, y)$ denote the basis and enrichment functions, respectively, u_i and α_j are coefficients (nodal values and generalized degrees of freedom, respectively), and s_j is a scaling coefficient introduced to avoid exceedingly sharp gradients in non-conforming elements for which the interface happens to pass very close to the nodes [50, 49].

The first term of this formulation represents the standard FEM portion of the approximation, while the second term represents the contribution of the enrichment functions in the solution field as applied by IGFEM. Unlike conventional GFEM/XFEM, the enrichment functions $\psi_j(x)$ are zero at the nodes of the non-conforming mesh, but assume nonzero values at the nodes defined by the intersection of the interface with the elements of the non-conforming mesh. Figure 3.2 presents a schematic view of the portion of the IGFEM solution constructed by the enrichment functions.

Figure 3.3 shows how the enrichment basis functions are defined for two types of domains: when an element is split by an interface into two quadrilaterals or when it is split into a

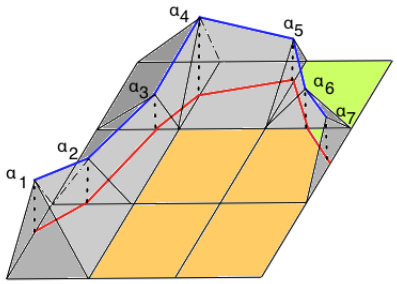


Figure 3.2: Contribution of enrichment function for modeling weak discontinuity in IGFEM. Generalized dof α_j denotes how enrichment functions are stitched together over interface nodes to provide continuous enrichment. No enrichment is attached to an interface node if it coincides with a node of original mesh.

triangle and a pentagon. They are defined as piecewise polynomials or tent functions.

As described in [50, 49], some of the key advantages of IGFEM, compared with conventional GFEM, include the straightforward fashion in which Dirichlet boundary conditions are applied, the reduced number of generalized degrees of freedom, and the ability to capture accurately problems characterized by highly mismatched material properties. Initial applications of IGFEM have shown great promise in the thermal analysis and computational design of microvascular materials [51] and the multiscale failure analysis of heterogeneous adhesives [8].

3.3.1 Hanging Nodes

In the context of pixel-aligned quad meshes, achieving the desired trade-off between solution accuracy and computational cost depends almost entirely on the quad mesh resolution. Thus, incorporating adaptive refinement into the method will help focus the computational resources on the material interfaces, thereby improving the overall accuracy of the solution. The main focus of refinement is of a geometrical nature: depending on the curvature of the interface inside each quad element traversed by that interface, higher- or lower-order IGFEM representations may be required. A crude finite element mesh will result in a crude segmentation of the different material phases. The most efficient solution will be achieved for meshes that are refined only when needed. Adaptive refinement creates hanging nodes in the mesh that must be dealt with appropriately. *A hanging node* is created on an edge

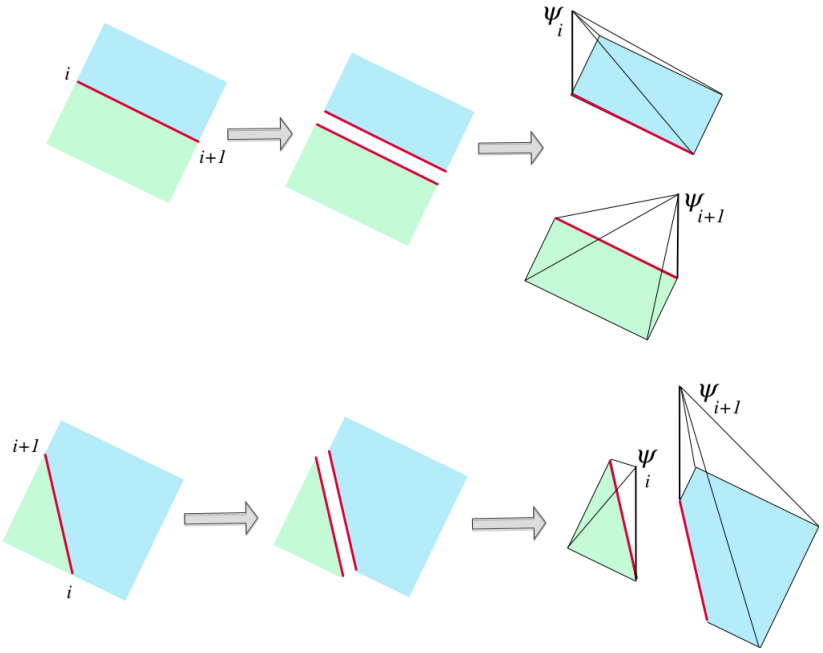


Figure 3.3: Basis functions applied at enrichment nodes in two types of elements created by interface-quad intersections.

when one of the surrounding elements is refined but not the element with the hanging node. The 1-irregular constraint [12] applied to the mesh guarantees at most one hanging node per edge and only in homogeneous elements that have no enrichment nodes present. Because of the recursive subdivision, hanging nodes can occur only in the middle of an edge. In dealing with hanging nodes in the finite element solution, we use the approach suggested in [28] for the extended-finite element method with hanging nodes, known as the constrained approximation.

Let us define I_h as the set of all hanging nodes in the mesh, and $I_r = I \setminus I_h$ as the set of all regular nodes, where set I is the set of all nodes. We also define Q_k as the pair of regular nodes associated associated with a hanging node,

$$Q_k = \{(i,j) : \text{nodes } (i,j) \in I_r \text{ sharing an edge with node } k\}, \quad k \in I_h.$$

Then, the nodal coefficient at the hanging node is computed as

$$u_k = 0.5u_i + 0.5u_j, \quad (i,j) \in Q_k.$$

In other words, we constrain the nodal coefficient at the hanging node to be the average of the regular nodes sharing an edge with it. When we compute the local (element) stiffness matrix, we treat the hanging node in the elements in which it is a corner node as a regular node, but keep track of it in the connectivity matrix. In the elements in which it is a hanging node indeed, we simply ignore it when we compute the local stiffness matrix. After we assemble the global stiffness matrix and the global load vector and solve for the nodal coefficients using the connectivity matrix, we replace it with the average of its regular node pair.

There are several other ways to deal with hanging nodes that could be explored, such as allowing degrees of freedom for the hanging nodes, but we leave this for future work.

3.3.2 Isoparametric Elements

A mesh can enable a higher degree of accuracy with a smaller number of elements by using higher order elements. Since the IGFEM mesh has only non-conforming elements, there will be no elements with complex shapes to deal with. The major complexity associated with higher order elements is that the domain of integration conforms to the curvature of the interface. However, there will be at most one curved edge per element.

The construction of shape functions over elements with curved boundaries becomes increasingly complicated as the geometry of the interface becomes more complex. This obstacle can be overcome by using isoparametric mapping from the physical coordinate system (x, y) to the local (ξ, η) element coordinates. The mapping is one-to-one and orientation preserving. The primary cost associated with this mapping is that of coordinate transformation.

The main properties of the mapping are:

- vertices are mapped onto vertices,
- midpoints of sides are mapped onto midpoints of sides,
- barycenters are mapped onto barycenters,
- degree of basis functions is preserved.

The isoparametric shape functions N_i^e at node i in element e are required to satisfy several conditions:

- have value 1 at node i and 0 at all other nodes,
- have C^0 continuity across element boundary,
- vanish over any element other than element e,
- satisfy the completeness condition: any displacement field that is a linear polynomial in x and y is represented exactly by interpolation.

In the case of quadrilateral domains of integration, ξ and η vary from -1 to 1, taking the value zero on the element median. This particular choice, rather than 0 to 1, has been introduced to simplify the Gauss quadrature computation of the integrals. For triangular domains of integration ξ and η vary from 0 to 1. Mappings from the triangular and rectangular domains to the local coordinates (ξ, η) are shown in Figure 3.4 and Figure 3.7, respectively.

The mapping is described by the following coordonate transformation of $x = s(\xi, \eta)$ and $y = t(\xi, \eta)$,

$$x = s(\xi,\eta) = \begin{bmatrix} N_1^e(\xi,\eta) & N_2^e(\xi,\eta) & \dots & N_i^e(\xi,\eta) & \dots & N_n^e(\xi,\eta) \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_i \\ \dots \\ x_n \end{bmatrix},$$

$$y = t(\xi,\eta) = \begin{bmatrix} N_1^e(\xi,\eta) & N_2^e(\xi,\eta) & \dots & N_i^e(\xi,\eta) & \dots & N_n^e(\xi,\eta) \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_i \\ \dots \\ y_n \end{bmatrix},$$

where (x_i, y_i) are the physical coordinates of the element nodes, N_i^e are C^0 continuous basis functions defined on canonical elements, and n is the number of shape functions associated with the isoparametric element. In the case of the bi-linear quadrilateral shown in Figure 3.7, n is 4. For the triangular element shown in Figure 3.4, n is 3.

The integrals used in computing the local (element) stiffness matrix requires computation of the area $dxdy$, which transformed in the local (ξ, η) element coordinates becomes:

$$dxdy = det(J)\, d\xi d\eta,$$

where J is the Jacobian matrix defined as

$$J = \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \xi} \\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta} \end{bmatrix}.$$

Since the shape functions N_i^e depend on ξ and η, the chain rule of differentiation is used to compute the derivatives of the shape functions

$$\frac{\partial N_i^e}{\partial \xi} = \frac{\partial N_i}{\partial x}\frac{\partial x}{\partial \xi} + \frac{\partial N_i}{\partial y}\frac{\partial y}{\partial \xi},$$

$$\frac{\partial N_i^e}{\partial \eta} = \frac{\partial N_i}{\partial x}\frac{\partial x}{\partial \eta} + \frac{\partial N_i}{\partial y}\frac{\partial y}{\partial \eta}.$$

Putting these two equations in matrix form, we can compute $\frac{\partial N_i}{\partial x}$ and $\frac{\partial N_i}{\partial y}$ making use of the inverse of the Jacobian,

$$\begin{bmatrix} \dfrac{\partial N_i}{\partial x} \\ \dfrac{\partial N_i}{\partial y} \end{bmatrix} = J^{-1} \begin{bmatrix} \dfrac{\partial N_i^e}{\partial \xi} \\ \dfrac{\partial N_i^e}{\partial \eta} \end{bmatrix}.$$

In our IGFEM implementation we allow the curvature of the interface to be approximated by polynomials of degree 1, 2 or 3. Although the mesh elements are all quadrilaterals, for integration purposes we use a combination of quadrilateral and triangular subdomains. Thus, the domains of integrations will be one of the following, based on the polynomial degree:

- bi-linear quadrilaterals
- linear triangles
- bi-quadratic quadrilaterals
- quadratic triangles
- bi-cubic quadrilaterals
- cubic triangles

For the triangular domains, we use Lagrangian elements, while for the higher-order quadrilaterals (bi-quadratic and bi-cubic) we use *serendipity elements* [60]. Serendipity elements, as opposed to Lagrangian elements, do not require additional internal nodes. Serendipity nodes rely solely on the corner nodes and although not as accurate as Lagrangian elements, they are more efficient (fewer shape functions) and avoid certain instabilities associated with Lagrangian elements. In fact, the L_2 error in interpolating the smooth function $u^h(x, y)$ by a piecewise bi-quadratic polynomial is $O(h^3)$, where h is the length of the longest edge of an element. Any additional degrees of freedom from a bi-cubic function do not (in general) improve the order of accuracy. Using a serendipity element with no interior nodes eliminates some of the shape functions and reduces the complexity of the approximation, while using cubic polynomials for element edges [60]. Once the shape functions are identified, the procedure for using the elements is similar to that for bi-linear quadrilaterals.

Below we illustrate the six domains of integrations we identified in our analysis, and the shape (basis) functions associated with the nodes of these elements.

Linear triangular elements

The three-node triangle is pictured in Figure 3.4. The corresponding shape functions are defined as

$$N_1^{e,1}(\xi, \eta) = 1 - \xi - \eta,$$
$$N_2^{e,1}(\xi, \eta) = \xi,$$
$$N_3^{e,1}\xi, \eta) = \eta.$$

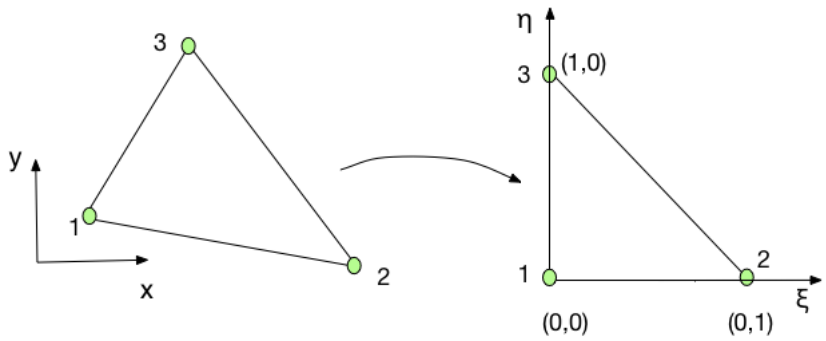


Figure 3.4: Linear triangular element mapped to isoparametric element.

These are the simplest piecewise-linear polynomials.

Quadratic triangular elements

To construct a quadratic polynomial, additional nodes must be defined at the midsides of the triangular element. This six-node triangle is pictured in Figure 3.5, and its corresponding shape functions are defined for

$$\begin{aligned}
\mathscr{G}_1(\xi,\eta) &= 1-\xi-\eta,\\
\mathscr{G}_2(\xi,\eta) &= \xi,\\
\mathscr{G}_3(\xi,\eta) &= \eta,
\end{aligned}$$

as

$$\begin{aligned}
N_1^{e,2}(\xi,\eta) &= 2\cdot\mathscr{G}_1(\xi,\eta)\cdot\left(\mathscr{G}_1(\xi,\eta)-\frac{1}{2}\right),\\
N_2^{e,2}(\xi,\eta) &= 2\cdot\mathscr{G}_2(\xi,\eta)\cdot\left(\mathscr{G}_2(\xi,\eta)-\frac{1}{2}\right),\\
N_3^{e,2}(\xi,\eta) &= 2\cdot\mathscr{G}_3(\xi,\eta)\cdot\left(\mathscr{G}_3(\xi,\eta)-\frac{1}{2}\right),\\
N_4^{e,2}(\xi,\eta) &= 4\cdot\mathscr{G}_1(\xi,\eta)\cdot\mathscr{G}_2(\xi,\eta),\\
N_5^{e,2}(\xi,\eta) &= 4\cdot\mathscr{G}_2(\xi,\eta)\cdot\mathscr{G}_3(\xi,\eta),\\
N_6^{e,2}(\xi,\eta) &= 4\cdot\mathscr{G}_3(\xi,\eta)\cdot\mathscr{G}_1(\xi,\eta).
\end{aligned}$$

Node 5, which in the general element corresponds to the interpolating point detected for

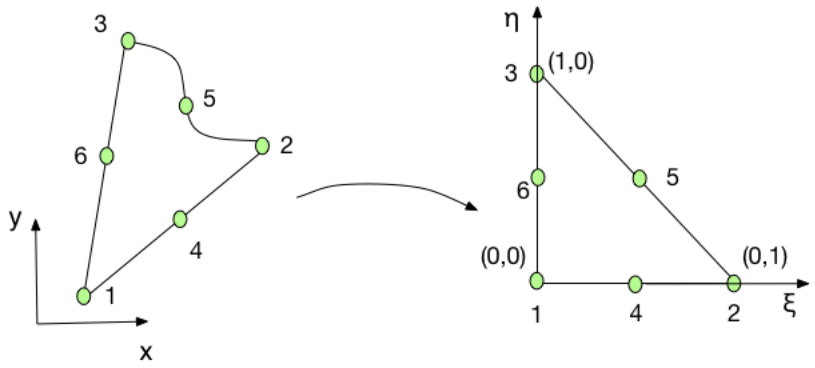


Figure 3.5: Quadratic triangular element mapped to isoparametric element.

quadratic interpolation may not lie exactly in the middle of the edge between points 2 and 3, as it is the case for the isoparametric element. Using this point between the two enrichment nodes (Node 2 and Node 3), we build the actual interpolating polynomial and then use this to determine the mid-point of the edge 2-3 in the general element that corresponds to Node 5 in the isoparametric element.

Cubic triangles

To build a complete cubic approximation we need 10 parameters or a ten-node triangle. Besides two additional points per each edge, this element also requires an interior point of the triangle. For the regular element, we take the interior point to be the centroid, which is guaranteed to lie inside the triangle. Similarly to the quadratic triangle, we determine the edge nodes 6 and 7 of the regular element by constructing a cubic polynomial with the points determined through image processing, and then evaluating this polynomial at intervals of $\frac{1}{3}$ on edge $2-3$. The cubic triangular element is shown in Figure 3.6. Using the same

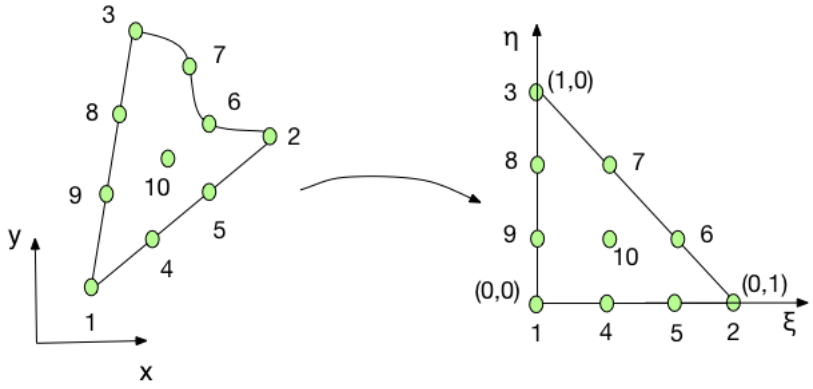


Figure 3.6: Cubic triangular element mapped to isoparametric element.

description of the $\mathscr{G}_{1,2,3}(\xi,\eta)$ functions, the corresponding basis functions are

$$
\begin{aligned}
N_1^{e,3}(\xi,\eta) &= \frac{1}{2}\cdot\mathscr{G}_1(\xi,\eta)\cdot\Big(3\mathscr{G}_1(\xi,\eta)-1\Big)\cdot\Big(3\mathscr{G}_1(\xi,\eta)-2\Big),\\
N_2^{e,3}(\xi,\eta) &= \frac{1}{2}\cdot\mathscr{G}_2(\xi,\eta)\cdot\Big(3\mathscr{G}_2(\xi,\eta)-1\Big)\cdot\Big(3\mathscr{G}_2(\xi,\eta)-2\Big),\\
N_3^{e,3}(\xi,\eta) &= \frac{1}{2}\cdot\mathscr{G}_3(\xi,\eta)\cdot\Big(3\mathscr{G}_3(\xi,\eta)-1\Big)\cdot\Big(3\mathscr{G}_3(\xi,\eta)-2\Big),\\
N_4^{e,3}(\xi,\eta) &= \frac{9}{2}\cdot\mathscr{G}_1(\xi,\eta)\cdot\mathscr{G}_2(\xi,\eta)\cdot\Big(3\mathscr{G}_1(\xi,\eta)-1\Big),\\
N_5^{e,3}(\xi,\eta) &= \frac{9}{2}\cdot\mathscr{G}_1(\xi,\eta)\cdot\mathscr{G}_2(\xi,\eta)\cdot\Big(3\mathscr{G}_2(\xi,\eta)-1\Big),\\
N_6^{e,3}(\xi,\eta) &= \frac{9}{2}\cdot\mathscr{G}_2(\xi,\eta)\cdot\mathscr{G}_3(\xi,\eta)\cdot\Big(3\mathscr{G}_2(\xi,\eta)-1\Big),\\
N_7^{e,3}(\xi,\eta) &= \frac{9}{2}\cdot\mathscr{G}_2(\xi,\eta)\cdot\mathscr{G}_3(\xi,\eta)\cdot\Big(3\mathscr{G}_3(\xi,\eta)-1\Big),\\
N_8^{e,3}(\xi,\eta) &= \frac{9}{2}\cdot\mathscr{G}_1(\xi,\eta)\cdot\mathscr{G}_3(\xi,\eta)\cdot\Big(3\mathscr{G}_3(\xi,\eta)-1\Big),\\
N_9^{e,3}(\xi,\eta) &= \frac{9}{2}\cdot\mathscr{G}_1(\xi,\eta)\cdot\mathscr{G}_3(\xi,\eta)\cdot\Big(3\mathscr{G}_1(\xi,\eta)-1\Big),\\
N_{10}^{e,3}(\xi,\eta) &= 27\cdot\mathscr{G}_1(\xi,\eta)\cdot\mathscr{G}_2(\xi,\eta)\cdot\mathscr{G}_3(\xi,\eta).
\end{aligned}
$$

Transforming a generic triangular element to the unit 45° right triangle we avoid the algebraic complexity associated with Lagrangian approximations, replacing it with that associated with transforming from one set of coordinates to another, which is more manageable. Once we transform to the canonical element, there is no need to transform back to the physical coordinates.

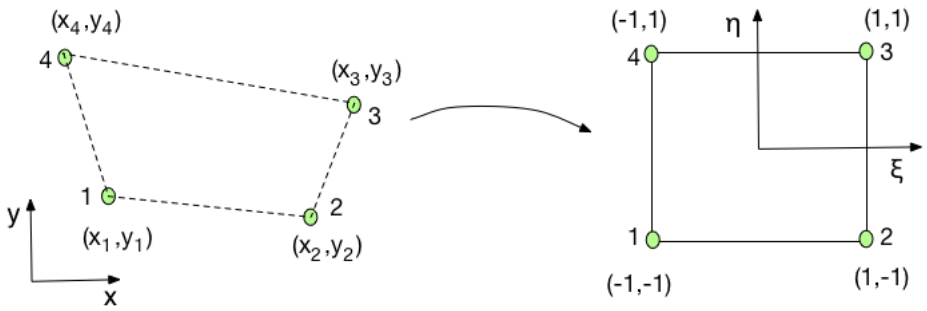


Figure 3.7: Four-node quadrilateral element mapped to isoparametric element.

Bi-linear quadrilateral elements

The simplest member of the quadrilateral family is the bi-linear element or the four-node quadrilateral, as shown in Figure 3.7. The shape functions over this isoparametric element are defined as

$$N_1^{e,1}(\xi,\eta) = \frac{1}{4}\cdot(1-\xi)\cdot(1-\eta),$$

$$N_2^{e,1}(\xi,\eta) = \frac{1}{4}\cdot(1+\xi)\cdot(1-\eta),$$

$$N_3^{e,1}(\xi,\eta) = \frac{1}{4}\cdot(1+\xi)\cdot(1+\eta),$$

$$N_4^{e,1}(\xi,\eta) = \frac{1}{4}\cdot(1-\xi)\cdot(1+\eta).$$

Bi-quadratic quadrilateral elements

The bi-quadratic quadrilateral is shown in Figure 3.8. The shape functions over this

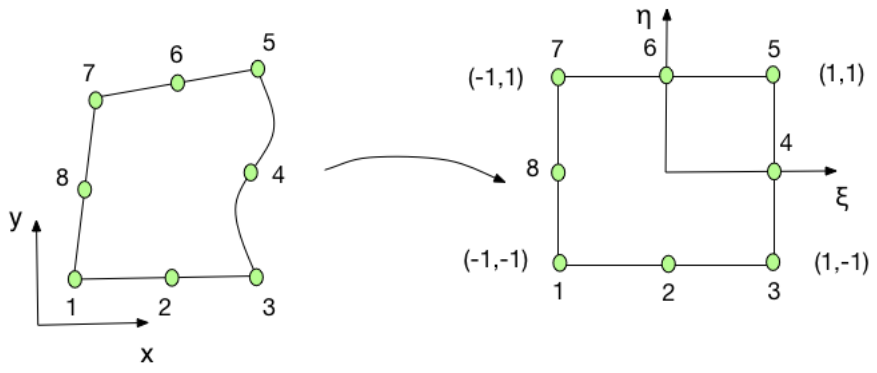


Figure 3.8: Eight-node quadrilateral element mapped to isoparametric element.

isoparametric element are defined as

$$
\begin{aligned}
N_1^{e,2}(\xi,\eta) &= \frac{1}{4}\cdot(1-\xi)\cdot(1-\eta)\cdot(-\xi-\eta-1),\\
N_2^{e,2}(\xi,\eta) &= \frac{1}{4}\cdot(1-\xi^2)\cdot(1-\eta),\\
N_3^{e,2}(\xi,\eta) &= \frac{1}{4}\cdot(1+\xi)\cdot(1-\eta)\cdot(\xi-\eta-1),\\
N_4^{e,2}(\xi,\eta) &= \frac{1}{4}\cdot(1+\xi)\cdot(1-\eta^2),\\
N_5^{e,2}(\xi,\eta) &= \frac{1}{4}\cdot(1+\xi)\cdot(1+\eta)\cdot(\xi+\eta-1),\\
N_6^{e,2}(\xi,\eta) &= \frac{1}{4}\cdot(1-\xi^2)\cdot(1+\eta),\\
N_7^{e,2}(\xi,\eta) &= \frac{1}{4}\cdot(1-\xi)\cdot(1+\eta)\cdot(-\xi+\eta-1),\\
N_8^{e,2}(\xi,\eta) &= \frac{1}{4}\cdot(1-\xi)\cdot(1-\eta^2).
\end{aligned}
$$

Bi-cubic quadrilateral elements

The bi-cubic quadrilateral is shown in Figure 3.9. The shape functions over this isoparametric element are defined based on whether they are a corner node or one of the two midside nodes.

For ξ_i and η_i coordinates of node i in the isoparametric element, one can define transformed coordinates

$$\hat{\xi}_i = \xi\xi_i, \quad \hat{\eta}_i = \eta\eta_i,$$

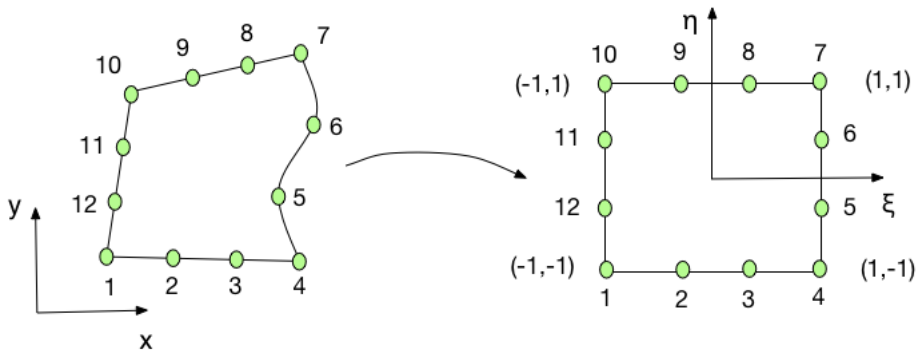


Figure 3.9: 12-node quadrilateral element mapped to isoparametric element.

corner node shape functions

$$N_i^{e,3} = \frac{1}{32} \cdot (1+\hat{\xi}_i) \cdot (1+\hat{\eta}_i) \cdot \big(-10 + 9 \cdot (\xi^2 + \eta^2\big), \quad i = 1, 4, 7, 10,$$

and midside node shape functions

$$N_i^{e,3} = \frac{1}{32} \cdot (1+\hat{\xi}_i) \cdot (1-\hat{\eta}_i^2) \cdot (1+9\hat{\eta}), \quad i = 5, 6, 11, 12,$$
$$N_i^{e,3} = \frac{1}{32} \cdot (1+\hat{\eta}_i) \cdot (1-\hat{\xi}_i^2) \cdot (1+9\hat{\xi}), \quad i = 2, 3, 8, 9.$$

3.3.3 Integration

When solving two-dimensional partial differential equations with the finite element method, computing the local stiffness and local load vectors usually requires computing integrals of the form

$$I = \int\int_{\mathscr{D}} F(x,y)\, dx\, dy,$$

where $\mathscr{D}$ is either a quadrilateral or triangular element. In the context of our thermal problem, after transforming to the isoparametric element, the integrand $F(x,y)dxdy$ is of the form

$$f(\xi,\eta)d\xi d\eta = \kappa\big(x(\xi,\eta), y(\xi,\eta)\big)\Big(\frac{\partial N_i(\xi,\eta)}{\partial \xi}\frac{\partial N_j(\xi,\eta)}{\partial \xi} + \frac{\partial N_i(\xi,\eta)}{\partial \eta}\frac{\partial N_j(\xi,\eta)}{\partial \eta}\Big)\, det\mathbf{J}_e\, d\xi\, d\eta,$$

where N_i, N_j are the basis functions, $\mathbf{J}$ is the Jacobian of the transformation, and $\kappa(x, y)$ is the conductivity of the material.

The integrals are computed numerically, using multi-dimensional Gaussian quadrature on the canonical elements. We use Gaussian integration because it can achieve the desired level of accuracy with a minimum number of points. This numerical method is the most efficient choice for computing element stiffness matrices, as the most time is spent in performing the integrations.

For the standard quadrilateral element $[-1, 1] \times [-1, 1]$, the Gaussian quadrature of order n with weights w_i and nodes u_i is

$$I = \int_{-1}^{1} \int_{-1}^{1} f(\xi, \eta) d\xi d\eta \approx \sum_{i=1}^{n} \sum_{j=1}^{n} w_i w_j f(u_i, u_j).$$

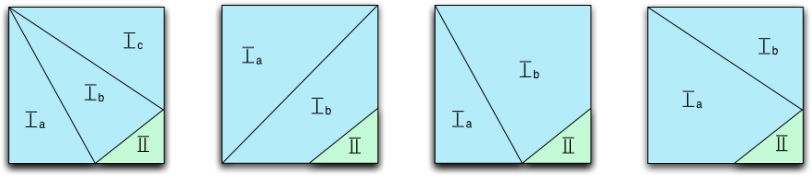
For the standard triangular element $(\xi, \eta) : 0 \leq \xi, \eta, \xi + \eta \leq 1$,

$$I = \int_0^1 \int_0^{1-\eta} f(\xi, \eta) d\xi d\eta = \int_0^1 \int_0^{1-\xi} f(\xi, \eta) d\eta d\xi = \frac{1}{2} \sum_{i=1}^{N} w_i f(u_i, u_i).$$

The number of integration points is chosen accordingly, to match the degree of the polynomials from the integrand. Since the integrand may contain rational terms in curved edge elements, exact integration may not be possible. Instead, the order of the quadrature is chosen as if the domain of integration were of straight edge elements, for which the Jacobian matrix is constant. The order should preserve the convergence rate estimates associated with the type of element.

For quadrilateral elements, the Gauss rule is a tensor product of one-dimensional Gauss formulas, while for triangular elements the rule must be constructed specifically for the triangular geometry. A four-point quadrature rule suffices for quadrilateral elements with up to cubic edges. For linear triangles we choose a 1-point quadrature, for quadratic triangles a 3-point rule, and for cubic triangles a 4-point rule.

The Gauss rule for triangles possesses triangular symmetry and positivity. The former means that for a sample point (u_i, u_j) present in the integration rule, the other points are obtained through all permutations of the two triangular coordinates. The latter means that



(a) (b) (c) (d)

Figure 3.10: Several ways domain of integration can be split.

all sample points lie inside or on the boundary of the triangle, and the weights are all positive.

3.3.4 Domains of Integration

The mesh contains only quadrilateral elements. However, a domain of integration could be either a quadrilateral or a triangle. A triangular domain could arise depending on how the interface intersects an element. In elements split by the interface, the integration computing the entries of the stiffness matrix must be done carefully, by requiring the elements to be divided into subdomains that conform to the material interface. Moreover, the conditioning of the local stiffness matrix is heavily influenced by how we choose to split the domain of the integration. In the case when the interface cuts the element into a triangle and a pentagon, the integration domain can be split such that one integral is computed over the triangle, while the other over the pentagon. To compute the latter accurately, the domain must be further subdivided.

Figure 3.10 shows the four possible ways the pentagon can be split: three triangles or variations of one triangle and a quadrilateral. In our implementation we chose the first option over the remaining three, due to its symmetry and to the fact that it avoids some of the problems of the other options. The second option can in some cases lead to a singular matrix when creating the basis functions. To see this, suppose the midpoint rule is being used. The three integration points are then co-linear, thus introducing the least amount of variability in coordinate directions of the domain. The last two options are similar to each other, and choosing one over the other would be arbitrary. Thus, to avoid singular matrices and random choices, we chose to proceed with the first option. Figure 3.11 shows how the

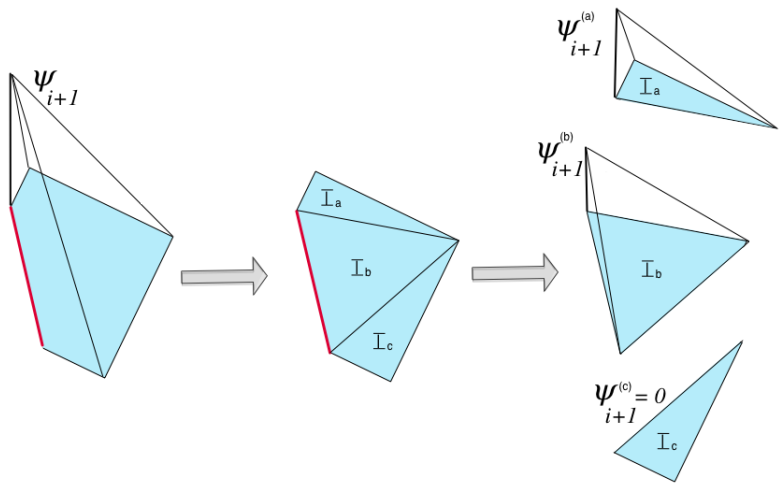


Figure 3.11: Enrichment basis functions.

enrichment basis function is defined in this subdomain: for each sub-triangle, the function is locally defined.

In the case when the interface cuts the element into two quadrilaterals, these shapes can be used as the domains of integration. In the elements without an interface inside, the whole element can be used as the domain of integration (there is no special handling of the domain).

The domains of integration can have not only straight-edge triangles or quadrilaterals, but can also have ones with curved edges. There are two approaches we could employ for curved edge domains, as shown in Figure 3.12, where the interface splits the domain into a curved triangle and a curved pentagon. One of the approaches is to subdivide further the domain with the curved edge, the pentagon, into multiple triangles (four), making use of **all** the interior nodes that determine the interface approximation. We could use these nodes to create a piecewise-linear approximation of the interface instead and then use linear isoparametric mapping of these regular elements to the unit 45° triangle, as shown in the bottom element of Figure 3.12. However, this approach would mean that we would change the conductivity of an area that could be of significant size (shaded in red in this element) to a different conductivity. The other approach and the one that we actually employ is to divide the pentagon into three triangles, and simply apply higher order (quadratic, cubic) isoparametric transformation, as illustrated in the top element in Figure 3.12.

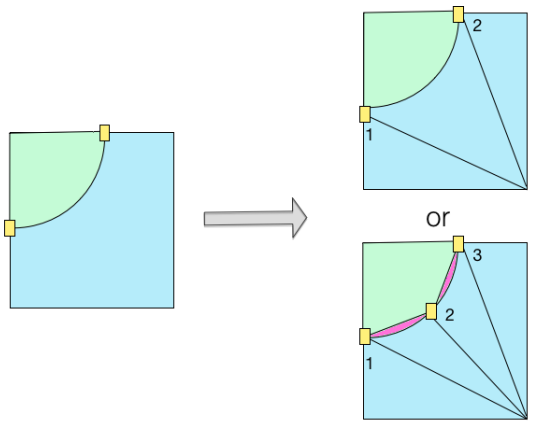
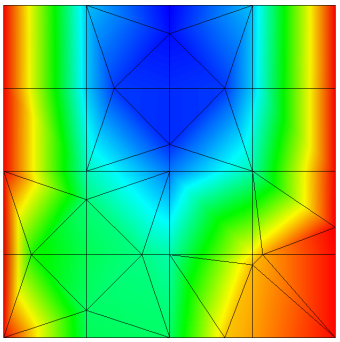


Figure 3.12: Curved interface creates additional challenges in integration.

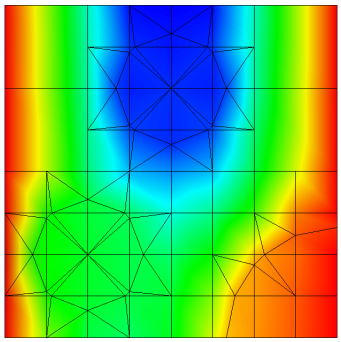
The various isoparametric mappings of the domains of integration depend on the degree of the interpolatory polynomial approximations to the interface. When linear polynomials are used, we have only 3-node triangles and 4-node quadrilaterals for the domain of integration. For quadratic polynomials interfaces we have the 6-node triangle and the 8-node quadrilateral. Similarly, for cubic interfaces, we have the 10-node triangle and the 12-node quadrilateral.

For various resolutions of the mesh, Figures 3.13 and 3.14 show the solution using the various domains of integration, based on the degree of the approximating polynomial for the Circles geometry shown in Figure 2.10a. Although the mesh appears to be a combination of triangular and quadrilateral elements, this appearance is deceiving. The mesh is truly quadrilateral, structured, and non-conforming, but for plotting purposes we show the solution over the domains of integration, rather than over the enriched quadrilateral elements, giving the appearance of a hybrid mesh. These figures illustrate well the interplay between h and p refinement. While there is no significant visual change in the overall IGFEM solution, from convergence studies we observe a slight improvement when higher order domains are used (Figure 3.16). The improvement is lost, however, when the mesh becomes finer and the need for q-refinement disappears.

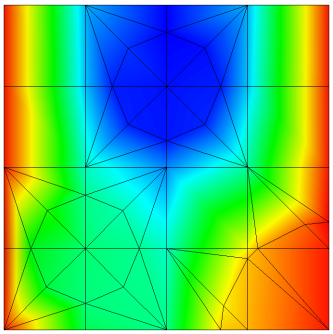
Table 3.1 provides statistical information about the meshes associated with these images. The first column gives the minimum and maximum element sizes; each of the remaining columns provides the total number of elements in the mesh and the number of heteroge-



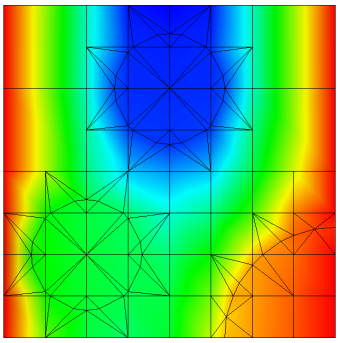
(a) Linears: MAX = 256 px, MIN = 256 px



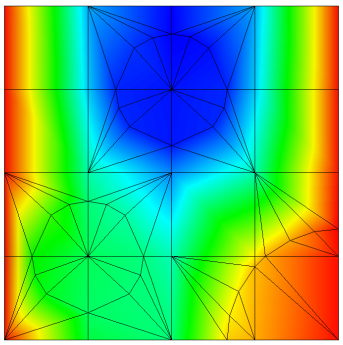
(b) Linears: MAX = 256 px, MIN = 128 px



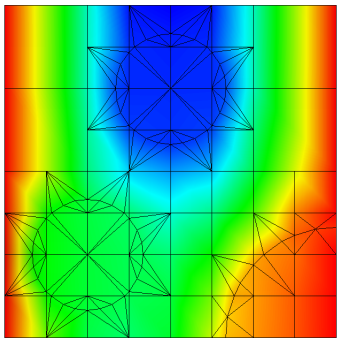
(c) Quadratics: MAX = 256 px, MIN = 256 px



(d) Quadratics: MAX = 256 px, MIN = 128 px

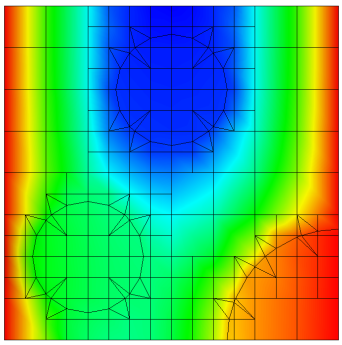


(e) Cubics: MAX = 256 px, MIN = 256 px

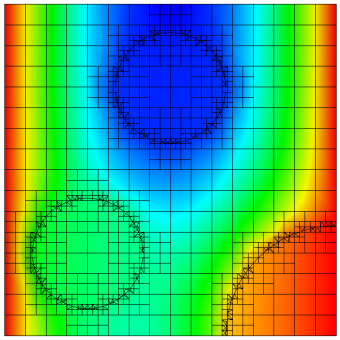


(f) Cubics: MAX = 256 px, MIN = 128 px

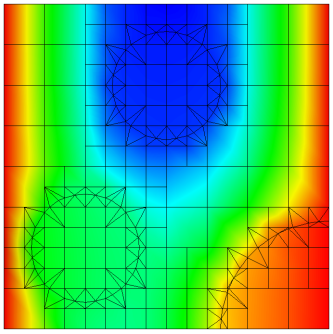
Figure 3.13: Comparison at various mesh resolutions of linear, quadratic and cubic polynomial approximations of interface. Maximum (MAX) element size is 256 px; minimum (MIN) element size is either 256 px or 128 px.



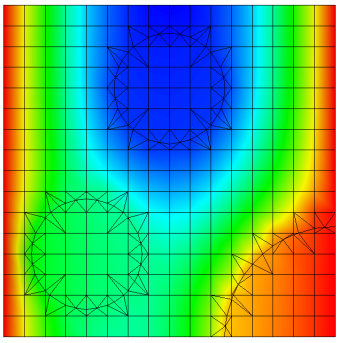
(a) Linears: MAX = 128 px, MIN = 64 px



(b) Linears: MAX = 64 px, MIN = 16 px



(c) Quadratics & Cubics: MAX = 128 px, MIN = 64 px



(d) Quadratics & Cubics: MAX = 64 px, MIN = 16 px

Figure 3.14: Comparison at various mesh resolutions of linear, quadratic and cubic polynomial approximations of interface. Maximum (MAX) element size is either 128 px or 64 px; minimum (MIN) element size is either 32 px or 16 px.

neous elements that had linear (L), quadratic (Q) or cubic (C) polynomial approximations triggered. For example, for a mesh resolution of MAX = 256, MIN = 128, we have a total of 52 elements when any of the three degrees (1, 2 and 3) of the polynomial are allowed. When we allow only linear approximations, out of the 52 elements, 21 are heterogeneous and have the interface approximated by a linear. When we allow quadratics, only one element can be approximated sufficiently well with a linear, the other 20 requiring a quadratic approximation. If we allow cubic approximations, four elements are well approximated by a quadratic, one by a linear, and the remaining 16 by a cubic. For a mesh resolution of MAX = 128, MIN = 64, cubics are not triggered. The same applies for the mesh with resolution MAX = 64, MIN = 8.

		Linear Approx.	Quadratic Approx.	Cubic Approx.
MAX = 256	Total # elements	16	16	16
MIN = 256	# refined elements	11 L	10 Q + 1 L	10 C + 0 Q + 1 L
MAX = 256	Total # elements	52	52	52
MIN = 128	# refined elements	21 L	20 Q + 1 L	16 C + 4 Q + 1 L
MAX = 128	Total # elements	175	175	175
MIN = 64	# refined elements	51 L	30 Q + 21 L	0 C + 30 Q + 21 L
MAX = 64	Total # elements	880	256	256
MIN = 8	# refined elements	293 L	30 Q + 21 L	0 C + 30 Q + 21 L

Table 3.1: Statistical information for meshes from Figures 3.13 and 3.14.

We observe that for coarser meshes there is a combination of elements with low-order and high-order domains of integration, while for finer meshes, the higher order domains are not used. In these finer meshes, the curvature of the interface in the smaller elements can be approximated well with low-order polynomials, and thus there is no need for 10-node triangles and 12-node quadrilaterals. For finer meshes we also observe the interplay between h- and q-refinements. For the mesh with MAX = 64, MIN = 8, allowing quadratics to interpolate the interface suffices and h-refinement is not triggered. Allowing only linears on the other hand triggers refinement and thus increases the total number of elements from 256 for quadratics to 880 for linears.

3.4 Heat-Conduction Test Problem

To test our 2D meshing methodology and implementation of IGFEM, we solve the following thermal problem using IGFEM with quadrilateral elements:

$$\begin{cases} -\nabla \cdot (\kappa(x,y)\nabla u) = f(x,y) & x,y \in \Omega = [0,1]\times[0,1], \\ u(x,y) = u_0, & x \in \Omega_D \text{ (Dirichlet)}, \\ \frac{\partial u(x,y)}{\partial y} = u_n, & y \in \Omega_N \text{ (Neumann)}, \end{cases} \tag{3.5}$$

where κ is thermal conductivity, u is temperature, and we apply appropriate Dirichlet and Neumann boundary conditions.

Multiplying this self-adjoint equation by a test function $v(x,y) \in H^1(\Omega)$ and then using integration by parts, we obtain

$$\int\int_{\Omega} \Big(\kappa \nabla u \cdot \nabla v\Big)\, dx\, dy = \int\int_{\Omega} f\, v\, dx\, dy + \int_{\Omega_N} \kappa\, u_n\, v\, ds.$$

In the discretized weak variational formulation we take the approximate solution space to be spanned by the basis functions N_i, and we choose the discretized approximate solution as given in the IGFEM formulation

$$u^h(\mathbf{x}) = \sum_{i=1}^{n} u_i N_i(\mathbf{x}) + \sum_{j=1}^{n_{en}} s_j\, \alpha_j \psi_j(\mathbf{x}),$$

where ψ_j are basis functions used at the enriched nodes. In our tests, they are the same as the N_i, although they need not be.

For an element defined by $y \in [a,b], x \in [c,d]$, the entries of its local stiffness matrix are given by

$$A_{i,j} = \int_a^b \int_c^d \kappa(x,y)\Big(\frac{\partial N_i}{\partial x}\frac{\partial N_j}{\partial x} + \frac{\partial N_i}{\partial y}\frac{\partial N_j}{\partial y}\Big)\, dx\, dy$$

and the entries of the load vector by

$$b_{i,j} = \int_a^b \int_c^d N_i(x,y) f(x,y)\, dx\, dy.$$

For isoparametric elements, these would become

$$A_{i,j} = \int_{-1}^{1} \int_{-1}^{1} \kappa(\xi,\eta) \Big(\frac{\partial N_i^e}{\partial \xi} \frac{\partial N_j^e}{\partial \xi} + \frac{\partial N_i^e}{\partial \eta} \frac{\partial N_j^e}{\partial \eta} \Big) \det(J)\, d\xi\, d\eta,$$

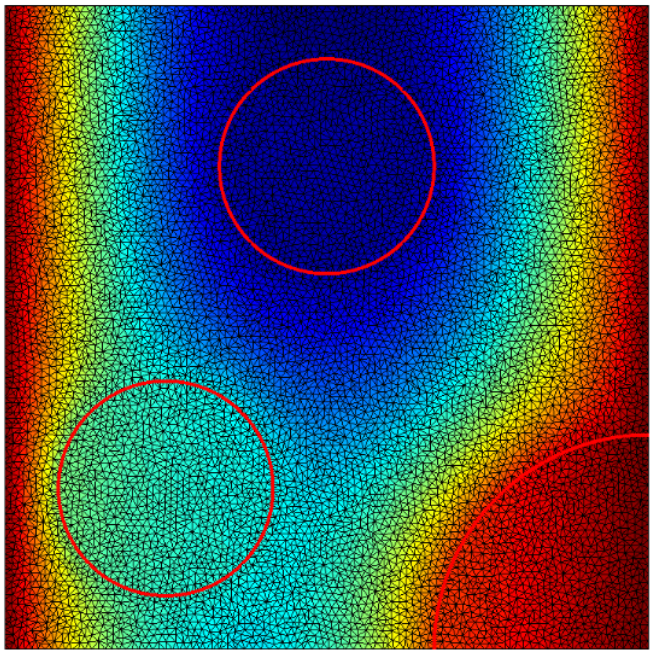
and

$$b_{i,j} = \int_{-1}^{1} \int_{-1}^{1} N_i^e(\xi,\eta) f(\xi,\eta) \det(J)\, d\xi\, d\eta.$$

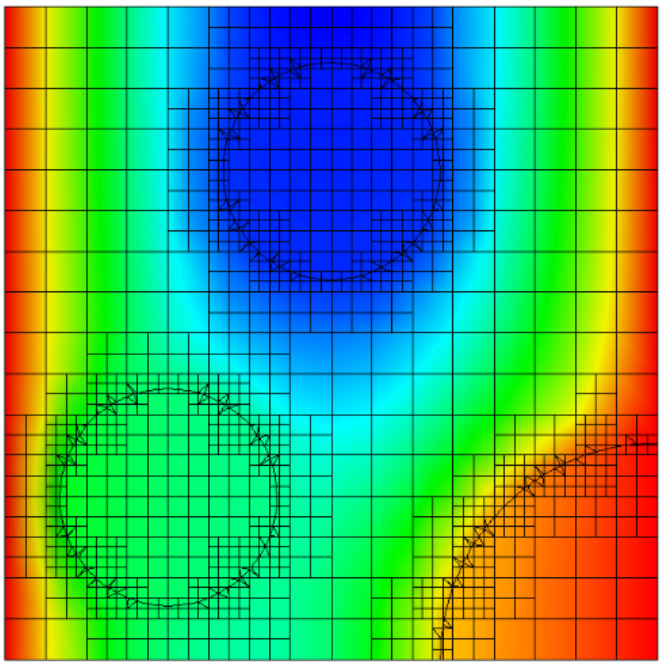
We then solve the resulting algebraic system $AU = b$ for the vector U of nodal values u_i and α_i. We store the matrix in the sparse format *COO* or *ijv*, where only row indices, column indices, and data entries of the matrix are stored. To solve the linear system we use the conjugate gradient method from *SciPy*.

We tested our implementation of IGFEM with quads using the heat conduction problem on a square domain with homogeneous Dirichlet boundary conditions on the left and right sides of the domain, and with zero flux applied at the top and bottom. The problem geometry is determined from the Circles image shown in Figure 2.10a, where the circular inclusions have different conductivities than the surrounding domain. Since this problem does not have a known analytical solution, we compared our solution to one obtained by using the classical finite element method with a highly refined conforming triangular mesh. This "benchmark" solution is shown in Figure 3.15a, along with a solution using IGFEM with quads in Figure 3.15b. The benchmark solution was used in computing the approximate L_2 norm of the error for solutions obtained by our code using IGFEM with quadrilateral elements. The L_2 error is plotted against mesh spacing for several q-refinements in Figure 3.16, where we observe nearly quadratic convergence.

We also compared the L_2 norm of the error with the number of elements in the mesh. For the regular finite element solution we generated several coarser triangular meshes that we compared with the finest triangular benchmark solution, and the results are shown in Figure



(a)



(b)

Figure 3.15: Finite element solutions for domain with circular inclusions having different conductivities than rest of domain. (a) Classical finite element solution with highly refined conforming triangle mesh. (b) Interface-based generalized finite element solution with nonconforming quadrilateral mesh.

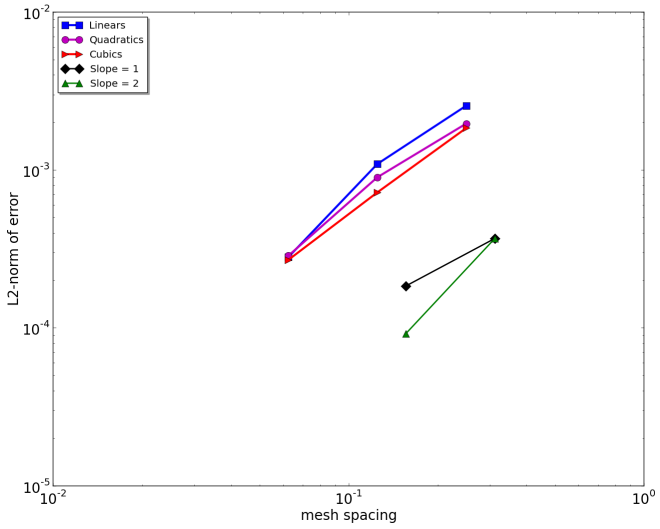
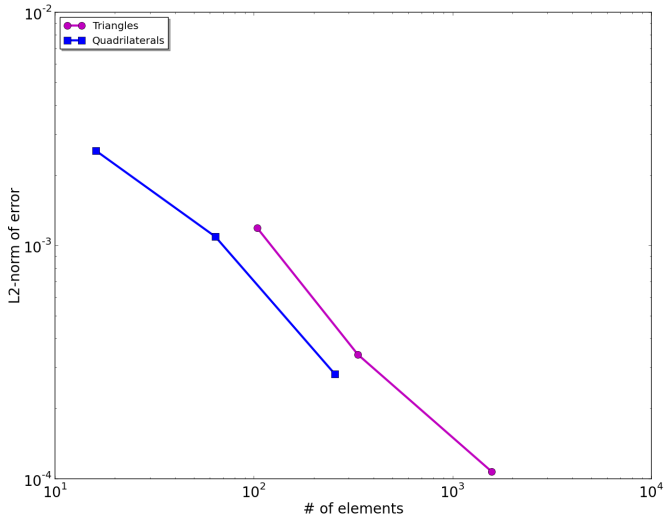


Figure 3.16: Convergence graph for geometry described in Figure 3.15b as compared to benchmark solution from Figure 3.15a.

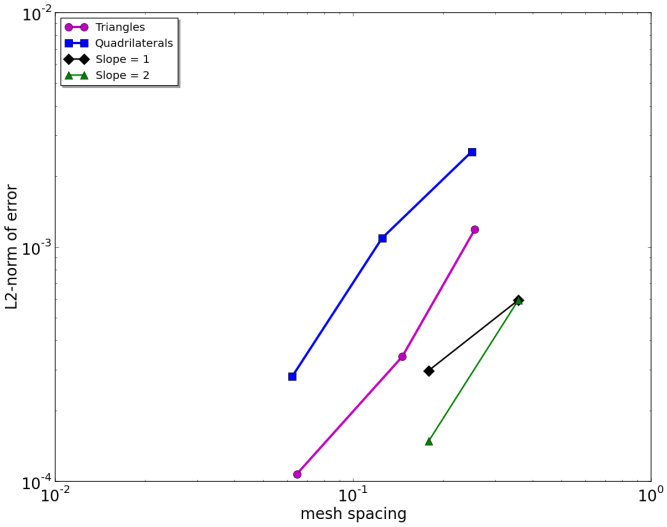
3.17a, where we see that the same level of accuracy can be achieved with fewer elements when using a structured quadrilateral mesh than when using a conforming triangular mesh. This observation can be reached another way from Figure 3.17b, where we plot the L_2 errors for both the quadrilateral mesh and the triangular mesh as a function of the mesh spacing. The smaller the spacing, the greater the number of elements in the mesh. The same level of accuracy is reached with coarser quadrilateral meshes than with finer triangular ones.

In Figures 3.18a and 3.19a we show the solution to the same thermal problem for the Channels and Horseshoe geometries shown in Figure 2.10b and 2.10c. The channels and the horseshoe-like inclusion have different conductivities than the rest of the domain. Figures 3.18b and 3.19b have the steep-gradient constraint applied: a minimum number of four homogeneous elements are required between interfaces.

We also solved the thermal problem on domains for which the geometry was deduced from real images containing noise, the Microvascular and SnBat shown in Figure 2.10d and 2.10g. Figure 3.20a shows the actual tomographic image of the microvascular material, while Figure 3.20b shows the IGFEM computed solution for this domain. Similarly, Figure 3.21a shows the actual tomographic image of the tin battery composite, while Figure 3.21b shows the IGFEM solution for it.



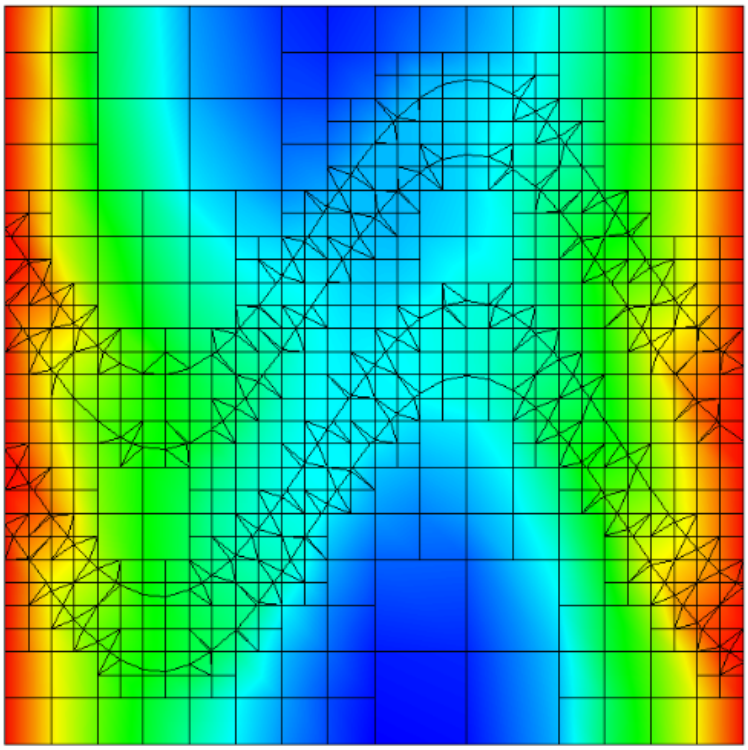
(a)



(b)

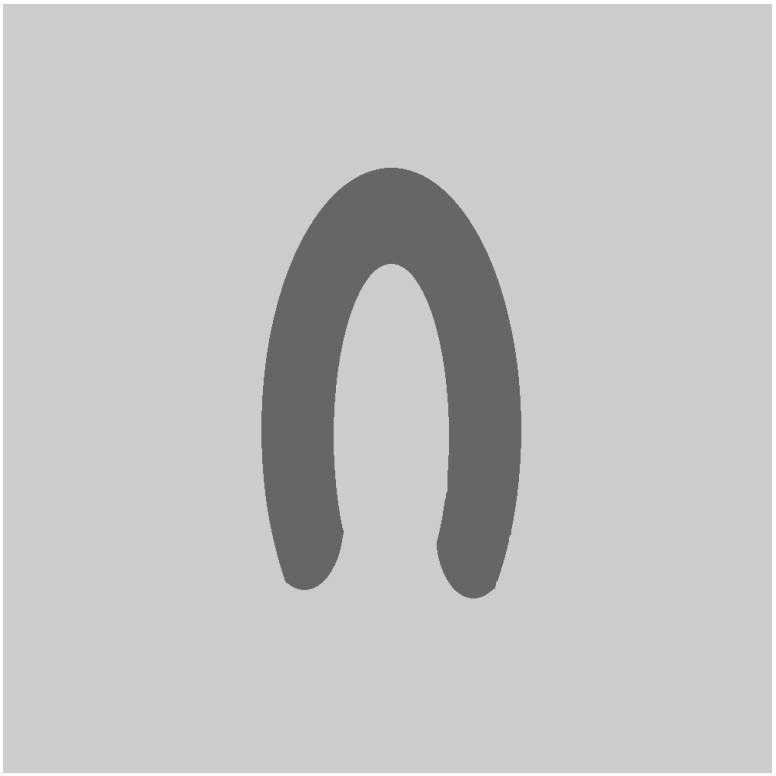
Figure 3.17: (a) Regular FEM solution (triangles) and IGFEM solution (quadrilaterals) as a function of number of elements in mesh. (b) Regular FEM solution (triangles) and IGFEM solution (quadrilaterals) as a function of mesh spacing.

(a)

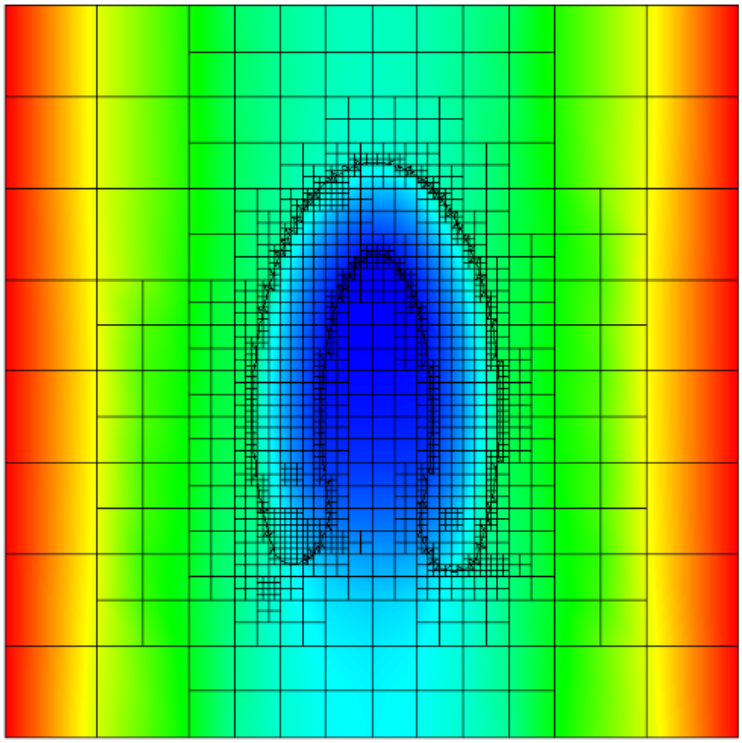


(b)

Figure 3.18: (a) Solution for two channels with conductivity different than rest of material. (b) Steep-gradient constraint applied to same domain as in (a).

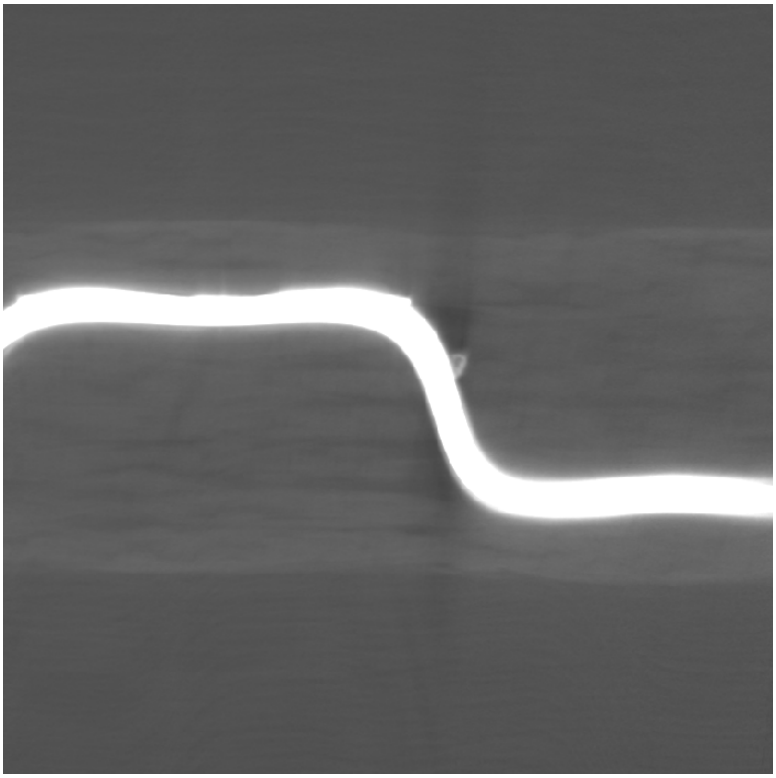


(a)

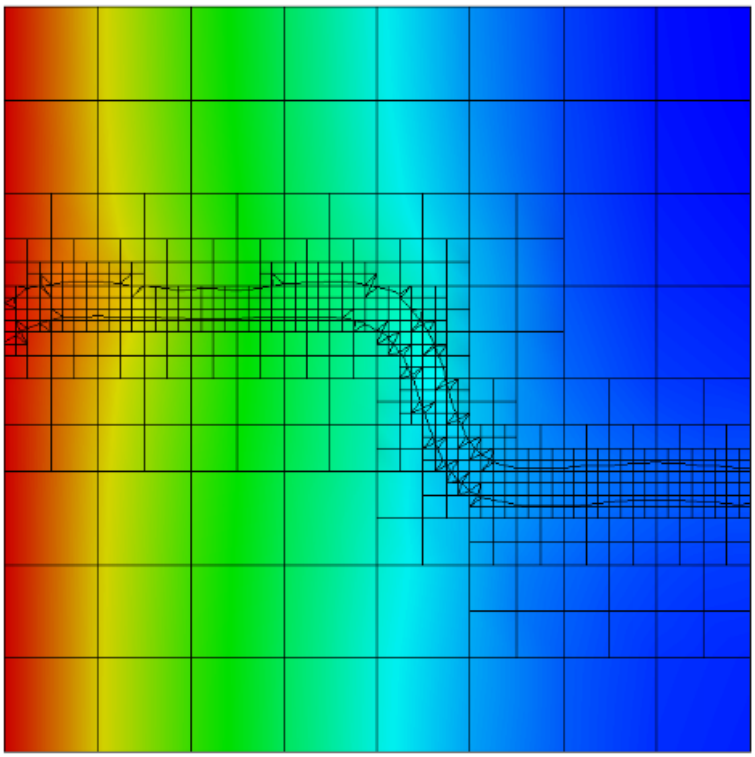


(b)

Figure 3.19: (a) Solution on horseshoe-like inclusion with conductivity different than rest of material. (b) Steep-gradient constraint applied to same domain as in (a).

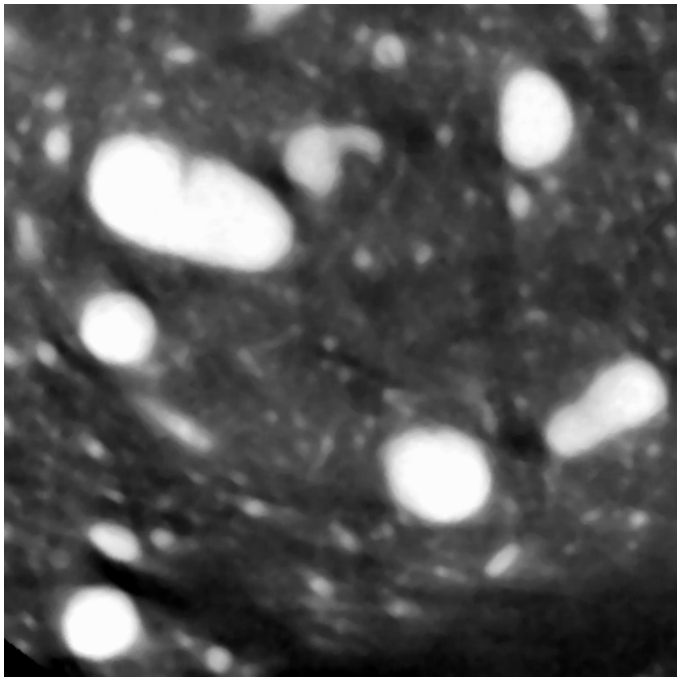


(a)

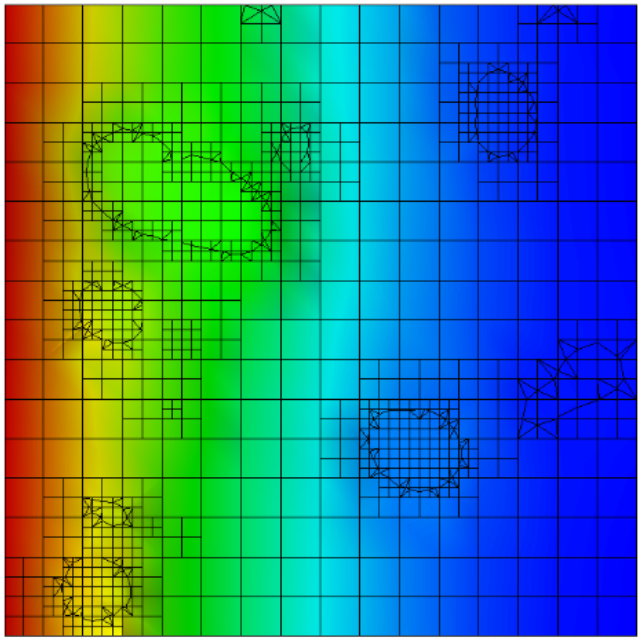


(b)

Figure 3.20: IGFEM applied to thermal problem for microvascular material. (a) shows X-ray image of material. (b) shows computed solution.



(a)



(b)

Figure 3.21: IGFEM applied to thermal problem for tin battery material. (a) shows X-ray image of battery. (b) shows computed solution.

CHAPTER 4

3D Meshing Based on Image Data

In this chapter we present a methodology for creating 3D meshes based on tomographic image data. Many of the concepts used in creating the 2D meshing algorithm presented in Chapter 2 remain the same. There are certain details that must be addressed, however, and we will do so in this chapter, referring back to sections of Chapter 2 as needed.

Three-dimensional meshes pose additional challenges and complexities that were not encountered in two dimensions. For example, approximating the interface surface in each element with a continuous function is a difficult problem that has been investigated in detail in the computer graphics literature [36, 38, 46]. The 3D meshing must satisfy not only the ordinary Finite Element Analysis constraints, but also those imposed by Interface-based Generalized Finite Element Analysis (IGFEM). The resulting mesh should be structured, non-conforming to the internal topology of the composite material, and relatively simple, as the complexity in dealing with composite materials should go into the 3D IGFEM implementation, rather than the mesh.

The remainder of this chapter is organized as follows. Section 4.1 discusses our octree-based meshing algorithm and presents the pseudo-code for it. Section 4.2 describes how the octree can be accessed, and Section 4.3 illustrates some of the 3D datasets we used in experiments. In Section 4.4 we present in detail how the IGFEM-imposed constraints on the mesh were adapted from 2D to 3D. Application of some of these constraints in 3D is straightforward, while others must either be modified or have additional constraints added. Section 4.5 discusses 3D interface detection and approximation. For the approximation we use either a planar least squares fit methodology described in Section 4.5.1, or a B-Spline (NURBS) interpolation described in Section 4.5.2. Each of these sections shows results when the octree algorithm is applied to various 3D voxel data sets.

4.1 Octree Algorithm

In 3D, structured meshes use hexahedrons, while unstructured meshes use tetrahedrons. In the context of tomographic data, we prefer hexahedrons as voxel-aligned meshes and voxel-defined interfaces allow for the FE mesh to be a subset of the underlying voxel mesh. We create hexahedral meshes using a recursive octree algorithm. Octrees are efficient for storing and retrieving data.

An octree generator is based on the same idea as a quadtree. The domain is enclosed by a cube (or more generally bricks), which is split into eight congruent subcubes. Each of these is then split recursively until the minimum sized cube intersects with the domain boundary. Any further splits are done based on a user-defined function or a balance condition. A balance condition for an octree could be that no cube is next to one very much smaller than itself. When we balance an unbalanced octree, the number of cubes is increased by a constant multiplicative factor.

Due to the approach we take for approximating interface surfaces inside hexahedral elements, we need not consider each case of element-interface-intersection individually, as we did in two dimensions. In three dimensions we use either a least squares method that fits a plane to a cloud of points, or a B-Spline interpolation that is applied only to faces of an element. We discuss these two approximations in detail in Section 4.5.

Implementation of the octree-based procedure is similar to that for quadtrees presented in Chapter 2. Algorithm 2 starts with a 3D voxel dataset as one large element, which is recursively subdivided until the maximum allowed element size is reached. For these elements, a test for homogeneity is done. If the test fails, then an interface approximation algorithm is tried until the minimum element size is reached. The subdivision is done recursively into eight octants as shown in Figure 4.1. On the left we show a hexahedral element and the directions of the axes. In image coordinates, $(0, 0, 0)$ starts in the back-top-left corner. The image on the right shows how we number the octree partitions. We refer to these as *partition 0*, *partition 1*, etc. in the octree algorithm described in Algorithm 2. Once this algorithm finishes building the tree structure, we employ additional techniques to modify the resulting octree to satisfy constraints on it imposed by the finite element method

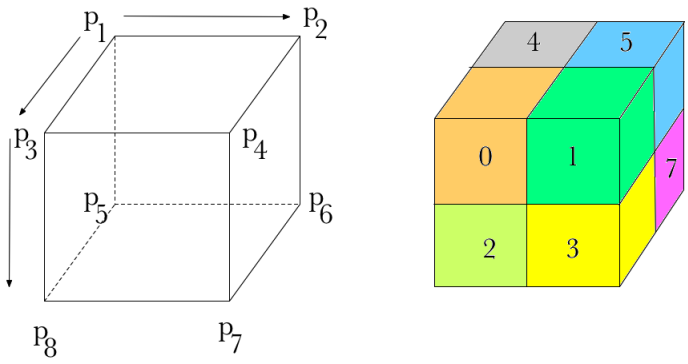


Figure 4.1: Image on left shows how 3D element corners are numbered, with $(0, 0, 0)$ coordinate starting in the back-top-left corner. Image on right shows octree partitioning.

of interest. These constraints are discussed in Section 4.4.

Test for Homogeneity

A given hexahedral element is *homogeneous* if there is no inclusion of significant size present in its interior. The element is *heterogeneous* otherwise. We determine whether the element is homogeneous or heterogeneous as we create the octree. In 2D the initial test was checking the pixel values of the four corners of the element. In 3D we check the voxel values of the eight corners. If they are all in the same bin of voxel values, then the element may be homogeneous, but an additional test is needed to verify whether there is indeed no inclusion of significant size inside it. For that purpose we use random sampling.

The probability of missing an inclusion of size $n \times n \times n$ voxels, after k uniform random trials in an $m \times m \times m$ voxels domain is given by

$$P = \left(\frac{m^3 - n^3}{m^3}\right)^k.$$

From this equation n can be determined for given P and k.

Algorithm 2 Generates mesh based on tomographic image

```
1: procedure OCTOFY(image, corners)
2:     if (element size ≥ MAX) then
3:         OCTOFY(image, partition 0 corners)
4:         OCTOFY(image, partition 1 corners)
5:         OCTOFY(image, partition 2 corners)
6:         OCTOFY(image, partition 3 corners)
7:         OCTOFY(image, partition 4 corners)
8:         OCTOFY(image, partition 5 corners)
9:         OCTOFY(image, partition 6 corners)
10:        OCTOFY(image, partition 7 corners)
11:    else
12:        if ( in_same_bin (all eight corners) ) then
13:            if (large_inclusion_present) then
14:                OCTOFY(image, partition 0 corners)
15:                OCTOFY(image, partition 1 corners)
16:                OCTOFY(image, partition 2 corners)
17:                OCTOFY(image, partition 3 corners)
18:                OCTOFY(image, partition 4 corners)
19:                OCTOFY(image, partition 5 corners)
20:                OCTOFY(image, partition 6 corners)
21:                OCTOFY(image, partition 7 corners)
22:            end if
23:        else
24:            for each element-interface intersection case do
25:                approximate interface with planar least squares fit or NURBS
26:                if (approximation "not good enough" & element size ≥ MIN) then
27:                    OCTOFY(image, partition 0 corners)
28:                    OCTOFY(image, partition 1 corners)
29:                    OCTOFY(image, partition 2 corners)
30:                    OCTOFY(image, partition 3 corners)
31:                    OCTOFY(image, partition 4 corners)
32:                    OCTOFY(image, partition 5 corners)
33:                    OCTOFY(image, partition 6 corners)
34:                    OCTOFY(image, partition 7 corners)
35:                end if
36:            end for
37:        end if
38:    end if
39: end procedure
```

4.2 Indexing and Computing Neighbors

As in 2D, we compute a unique index or location code for each element. The index describes the path from the root to the element in the octree. We compute the location codes using Morton numbering sequence and bit-interleaving [42, 22]. For indexing, we assume a uniform partition in all three dimensions, x, y and z. For each cell of the uniform partition there is an (i, j, k) index associated with it. Each of these indices is converted to binary. After bit interleaving, we obtain a number in base 2 that is then converted to base 8. The reason for choosing base 8 is because we have eight partitions per element.

The Morton number M for a triple of coordinates in base 10 (X_{10}, Y_{10}, Z_{10}), converted in base 2 to (X_2, Y_2, Z_2), is given by the one-to-one function

$$M = \frac{\sum_{i=0}^{T-1}(2^{2i} * x_i + 2^{3i+1} * y_i + 2^{3i+2} * z_i)}{2^3},$$

where T is the maximum length of the string representation of X_2, Y_2 and Z_2, and x_i, y_i, z_i represent the bit indices of X_2, Y_2 and Z_2, respectively.

Once we have a location code associated with each tree node, we can use the generalization to 3D of the Finite-State-Machine (FSM) algorithm discussed in Chapter 2 [58]. Given an index node and a direction, we can use Table 4.1, which shows a reduced version of the FSM algorithm, to compute the index of a neighbor in the required direction. Computing the orthogonal directions {D, U, R, L, B, F} requires only a lookup in the table. The twelve edge neighbor directions {RD, RU, BL, etc.} require two lookups, while the eight corner directions {ULF, ULB, etc.} require three lookups.

Each lookup is relatively fast; in the worst case it requires a full n lookups, where n is the depth of the tree. The worst case is also encountered when a node fails to exist in the tree. In that case, the tree will have either the parent or a child of the node. An additional query into the tree is then required to determine which of these is the true neighbor.

Direction	0	1	2	3	4	5	6	7
D	2, H	3, H	0, D	1, D	6, H	7, H	4, D	5, D
U	2, U	3, U	0, H	1, H	6, U	7, U	4, H	5, H
R	1, H	0, R	3, H	2, R	5, H	4, R	7, H	6, R
L	1, L	0, H	3, L	2, H	5, L	4, H	7, L	6, H
B	4, H	5, H	6, H	7, H	0, B	1, B	2, B	3, B
F	4, F	5, F	6, F	7, F	0, H	1, H	2, H	3, H

Table 4.1: Reduced Finite-State-Machine lookup table for finding neighbors in octree [58]. D - for down, U - for up, R - for right, L - for left, B - for back, F - for front, H - for halt.

4.3 Test Images

We use the datasets pictured in Figures 4.2, 4.3, 4.4, and 4.5 to illustrate some of the meshing techniques described in the following sections of this chapter. Some of these datasets had 2D slices extracted from them and used in Chapter 2 to illustrate 2D meshing.

- Figure 4.2 shows the interface surfaces of an artificially created dataset of size 512 $\times$ 512 $\times$ 512 voxels. The width of each channel is about 120 voxels.

- Figure 4.3 is an artificial dataset created to mimic composite materials with fibers embedded inside. The set is also of size 512 $\times$ 512 $\times$ 512 voxels. The interior fibers are of various diameters ranging from about 60 voxels to 110 voxels.

- Figure 4.4 shows the interface surfaces of a tin battery electrode material. Its size is 384 $\times$ 384 $\times$ 384 voxels. To allow for the relatively high level of noise in the raw data, we first applied smoothing followed by conversion to only two grey levels, so that the material interfaces are sharply defined.

- Figure 4.5 displays the interface surface of a microvascular channel embedded in a polymer. The image is of size 400 $\times$ 400 $\times$ 500 voxels, with the largest diameter of the channel of about 30 voxels. For this image we used only a threshold to delineate the interface from the polymer, as no smoothing was needed.

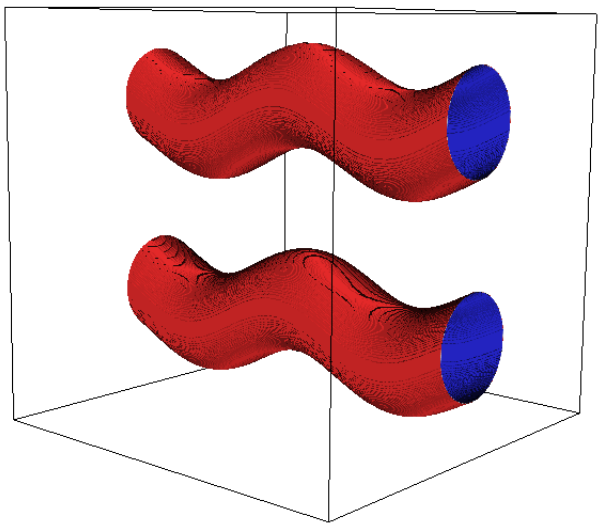


Figure 4.2: Channels: interface surfaces of manufactured channels.

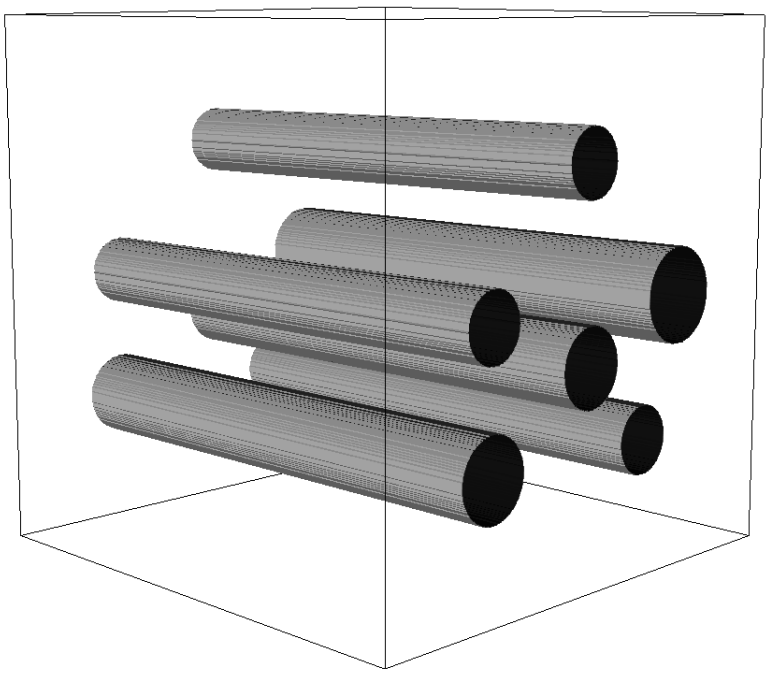


Figure 4.3: Fibers: interface surfaces of manufactured fibers.

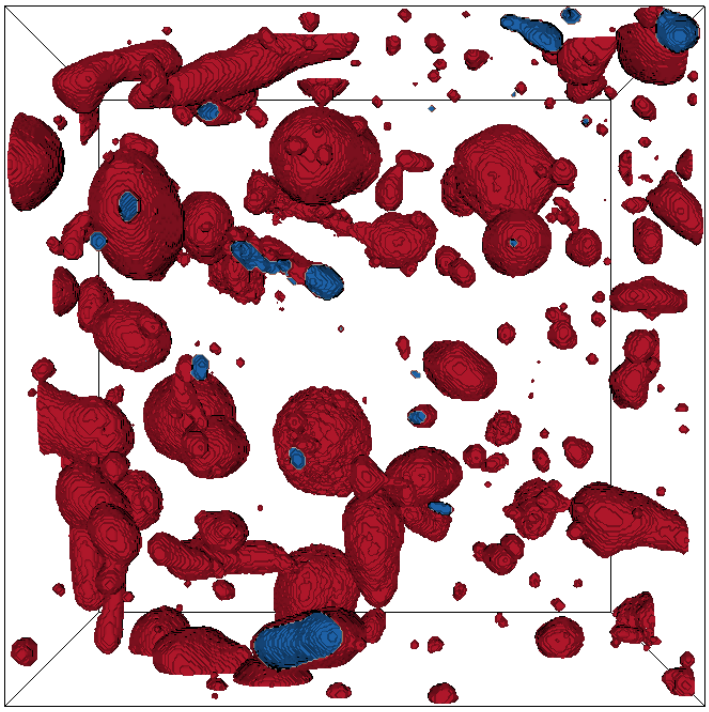


Figure 4.4: SnBat: interface surfaces of tin battery

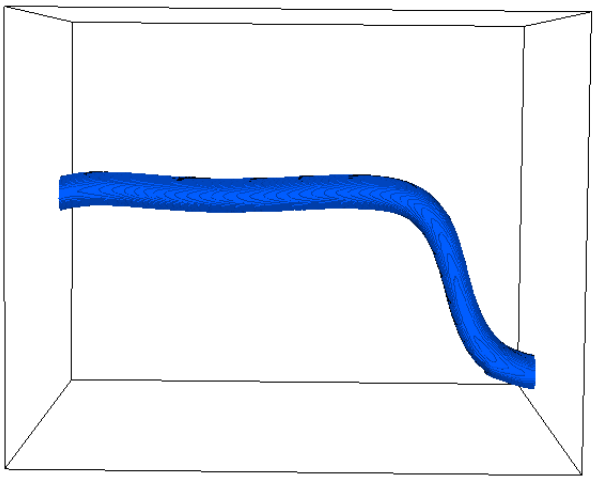


Figure 4.5: Microvascular: interface surface of partial microvascular channel.

4.4 IGFEM Imposed Constraints

In 3D we consider the same IGFEM-imposed constraints discussed in Chapter 2. Many of these constraints remain the same in higher dimensions with a few exceptions and various modifications for the additional dimension. We will discuss each constraint individually.

IGFEM-imposed constraints

1. h-refinement,
2. interface constraint,
3. 1-irregular rule or tree rebalance,
4. k-neighbor rule,
5. steep-gradient constraint.

4.4.1 Mesh Size Constraint

The aim of adaptive finite element methods is to generate effective meshes that deliver an accurate solution with a smaller computational cost. High accuracy can be captured with dense meshes, but at high computational cost. Alternatively, non-uniformly refined meshes can achieve good accuracy using relatively coarse meshes, resulting in lower costs.

Using the same definition of h-refinement as in Chapter 2, we recursively refine the mesh until the interface can be adequately approximated by a continuous function of predetermined degree: a planar least squares fit or a B-Spline surface interpolation, both discussed in Section 4.5. As opposed to the 2D meshing algorithm, in 3D we do not do q- and hq- refinements, as we allow only one type of surface approximation or interpolation. These refinements could be investigated in future work.

The h-refinement retains some of the properties from 2D meshing

- mesh is kept as coarse as possible, while still enabling the require solution accuracy,
- mesh is structured,
- mesh is non-conforming.

4.4.2 Interface Constraint

In 3D, a *hanging node* or an *irregular node* is defined as a node where the edges of a refined element meet at the midsides or center of a face of a coarser one.The last part of this definition, a *node on a center of a face*, is specific to three dimensions and must be considered when discussing the application of IGFEM constraints in more than two dimensions.

As in 2D, hanging nodes are created when refinement is triggered locally without propagating to surrounding elements. Elements crossed by an interface and with surrouding finer elements create a situation where both hanging nodes and enrichment nodes are present. This is undesirable, as the IGFEM formulation must deal not only with enrichment nodes in a heterogeneous element, but also with the irregular nodes. To simplify the mathematics, we require that no heterogeneous element should have both types of nodes present on its edges or faces.

The procedure for applying this constraint is similar to that described in Chapter 2. For each heterogeneous element we consider all neighboring elements with which a face or an edge is shared, 6 face-neighbors and 12 edge-neighbors. If any of these neighbors is finer than the current element (meaning a hanging node is present), we trigger subdivision. We do not have to check all 18 neighbors: as soon as we find one refined, we stop the search.

4.4.3 1-irregular Rule

As we go to three dimensions, the size of the octree increases, making it more expensive to traverse. This cost can be ameliorated by reducing the number of levels in the tree one would have to go through to find other nodes of the tree. This is achieved through tree rebalancing (or 1-irregular rule), by forcing at most one level difference between adjacent nodes. The rule staes that *"on any edge or in the interior of any face there can be no more than one irregular node"* [41]. The rebalancing is done *a-posteriori* based on a breadth-first traversal, starting from the root.

As pointed out in [6, 43] conforming mesh refinement in the classical finite element method is more complex in 3D than just generating 1-irregular meshes with hanging nodes. This makes 1-irregular meshes more appealing than uniformly refined ones.

The procedure is similar to that described in two dimensions (Chapter 2). The previous constraint guarantees that no heterogeneous element will have an adjacent neighbor that is refined, meaning the 1-irregular rule will be applied to homogeneous elements only. For each node in the tree representing a homogeneous element in the mesh, we look at the adjacent neighbors with which a face or edge is shared (18 total), and check whether these nodes are refined. If they are, then we look at its eight children and see if they are refined. If the children with which the face or edge is shared are refined, then subdivision is triggered in the current element. Again, as soon as we find one neighbor with grandchildren, we subdivide; there is no need to consider all the neighbors.

4.4.4 k-neighbor Rule

In 2D, the *k-neighbor* rule is known as the 3-*neighbor rule*, which requires that no three or more edge neighbors be refined. In 3D, this rule is split into two parts: a k_1-*neighbor rule* and a k_2-*neighbor rule*. The k_1 rule is a direct generalization of the 3-neighbor rule in 2D, while the k_2 rule is specific to 3D.

We use the variant of the k-neighbor rule proposed by Moore [41]. An element is refined if k_1 of its face neighbors are refined or if it has k_2 or more nearly regular edges. An edge is *nearly regular* if the two face-neighbors and one edge-neighbor associated with this edge are refined. We use the values suggested by Moore for the two parameters, $k_1 = 5$ and $k_2 = 2$, but these values can be changed. In this constraint, as in the others, as soon as one of the rules has at at least one element violating the requirements, subdivision is triggered.

4.4.5 Steep-gradient Constraint

The idea behind the steep-gradient constraint in 3D remains roughly the same as in 2D. To guarantee that there is a minimum number of homogeneous elements between two interfaces, whether those are part of the same inclusion or of different ones, we allow an additional minimum size that is alternative to the minimum size pre-set for the mesh spacing. For each heterogeneous element we build the normal vector to the interface inside and step in

that direction across elements. As we go along, a count is kept of how many homogeneous elements are crossed before a heterogeneous one is encountered. The number of required homogeneous elements between interfaces is a parameter that can be changed.

As opposed to 2D, not only face neighbors must be stepped through, but also corner neighbors, those with which only a node is shared. This translates into an additional direction to search (a three-letter direction in the finite state machine table, versus two letters or one).

4.4.6 Final Constraints

Without the last constraint applied, the steep-gradient constraint, the mesh is ready to use with the interface-based finite element method. However, the refinements created by that constraint are all local, and the resulting octree may violate some of the previous constraints, such as the 1-irregular and k-neighbor rules. Thus a final pass through all of the previous constraints must be made after applying the steep-gradient constraint before the mesh is ready for use.

Figures 4.6, 4.7, 4.8 and 4.9 show three-slices of these data sets meshed with all constraints applied. For Figure 4.9, the mesh is relatively coarse outside the channel. Refinements are triggered only in and near the channel, so that some of the three-slices intersect the coarse elements inside them rather than on their faces, as in the case of the other figures.

Tables 4.2, 4.3 and 4.4 show statistical information about the mesh size and the contribution of each constraint to the final mesh. The Fibers dataset (Table 4.2) sees a dramatic increase in number of elements after the steep-gradient constraint is applied, with the total number of elements after all constraints being almost two orders of magnitude larger. The contribution of the gradient constraint is so large due to the fact that the fibers, although with a simple geometry and little curvature, are very close to each other. With a relatively coarse mesh, that closeness violates the requirement that a minimum number of homogeneous elements lie between interfaces.

In the case of the tin battery (Table 4.3), the interface constraint triples the size of the mesh, while the 1-irregular rule and the k-neighbor rule have hardly any influence. The steep-gradient constraint contributes relatively few more elements to the mesh, but the

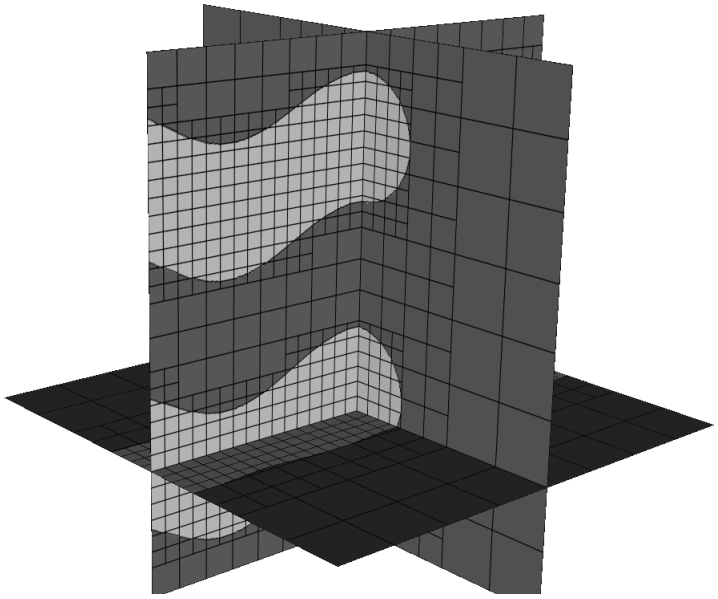


Figure 4.6: Channels with all constraints applied.

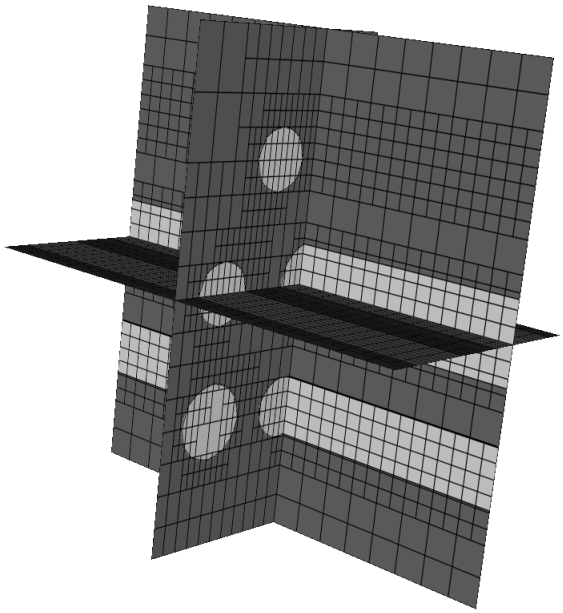


Figure 4.7: Fibers with all constraints applied.

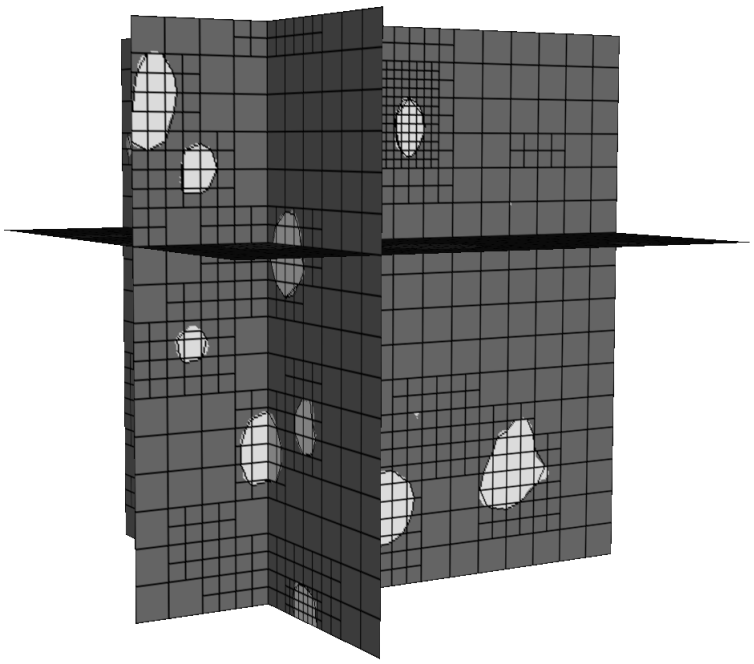


Figure 4.8: Tin battery with all constraints applied. Planar least squares fit approximation was used.

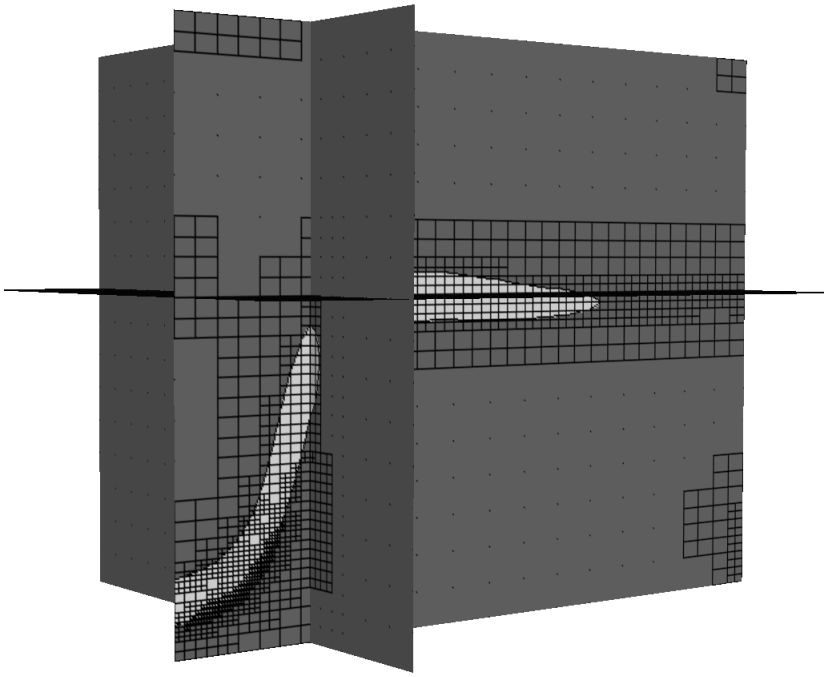


Figure 4.9: Microvascular channel with all constraints applied. Planar least squares fit approximation was used. Large blank areas represent elements larger that those shown, as channel is not orthogonal to x, y or z directions.

cumulative effect of those elements increases the final mesh to over six times the mesh size after the initial constraint.

The meshes for the microvascular channels (Table 4.4) sees the least impact from the IGFEM constraints. The interface constraint has the most significant impact of all the constraints, as it almost doubles the mesh. The remaining constraints have a very small effect on the overall mesh size. Since the channels have a relatively large diameter and are relatively far apart, the steep-gradient constraint barely triggers any new subdivisions. The new ones that are triggered, however, have the effect of doubling the mesh size after all constraints are applied.

Constraints	Planar		NURBS	
	total elems	CF	total elems	CF
h-refinement	281	1	176	1
interface constraint	1716	6.1	428	2.4
1-irregular rule	1800	6.4	428	2.4
k-neighbor rule	1800	6.4	512	2.9
gradient constraint	10662	37.9	1961	11.1
all constraints	23668	84.2	10984	61.9

Table 4.2: Fibers: statistical information about mesh size and cumulative increase (CF) after each IGFEM-imposed constraint on mesh.

Constraints	Planar		NURBS	
	total elems	CF	total elems	CF
h-refinement	1730	1	1716	1
interface constraint	5262	3	5202	3
1-irregular rule	6430	3.7	6397	3.7
k-neighbor rule	7264	4.2	7234	4.2
gradient constraint	7341	4.2	7311	4.3
all constraints	10970	6.3	10853	6.3

Table 4.3: Tin battery: statistical information about mesh size and cumulative increase (CF) after each IGFEM-imposed constraint on mesh.

Constraints	Planar		NURBS	
	total elems	CF	total elems	CF
h-refinement	4412	1	4251	1
interface constraint	7761	1.8	7752	1.8
1-irregular rule	9038	2.1	9038	2.1
k-neighbor rule	9332	2.1	9332	2.2
gradient constraint	9612	2.2	9612	2.3
all constraints	16514	3.8	16514	3.9

Table 4.4: Microvascular channels: statistical information about mesh size and cumulative increase (CF) after each IGFEM-imposed constraint on mesh.

4.5 Localization and Approximation of Material Interfaces

Locating and representing an interface in 3D becomes slightly more complicated than in 2D. The main idea, however, remains the same. We employ two methods to determine the curvature of the surface: one using a least squares fit, and one using B-Spline interpolation.

The interface is localized first on faces of an element by searching along each edge of an element and determining the intersection points. We also shoot rays through an element to determine interior points that we then use in better determining the shape of the surface inside the element.

Surface Continuity

As in 2D, we do not build a single function to represent the surface globally. Each interface approximation is done locally, within each element. Therefore, we enforce only C^0 continuity by using the same intersection points on edges of adjacent elements. Enforcing C^0 continuity also helps in reducing the redundancy in determining intersection points: we search for a change in material properties (i.e., voxel values) only along edges that have not been searched previously.

4.5.1 Planar Surface Approximation

In 2D we approximated the interface with Newton polynomials of degree one, two, and three and NURBS of order three that interpolated data points determined by shooting rays

through an element and finding intersection points of the rays with the interface. In 3D, we use two approaches to determine the interface and its curvature: an approximation method, and an interpolation method. For the approximation, we use a planar least squares fit, discussed in this section, while for the interpolation we use 3D NURBS, discussed in the next section.

The procedure for planar least squares fit is as follows.

- Along each edge of an element, look for intersections with the interface. Add these points to a list of intersection points. We allow at most one intersection point per edge; otherwise we require the element to be subdivided.
- For all the points in the list of intersection nodes, do a planar least squares fit, and compute the residual.
- If the residual is greater than a pre-determined tolerance, a planar fit does not do a good enough job and the element will be subdivided. If the residual is smaller than the tolerance, we conclude that we have found a good fit for the surface with a plane. Notice that it does not matter *why* the planar fit fails, as a curved interface or multiple interfaces should trigger refinement in either case.

We illustrate this process in Figure 4.10 where the interface cuts an element in two, through four intersection nodes shown in red in part (a) of the figure. In part (b) we show a planar least squares fit to these points. In part (c) we show both the original interface curves on each face, and the planar approximation.

Figures 4.8, 4.9 and 4.11 illustrate planar least squares fit approximation. A planar fit is relatively simple to implement, but may require substantial refinement to achieve desired accuracy.

4.5.2 Non-Rational Uniform B-Splines in 3D

Another way the interface can be represented is through surface interpolation, achieved through B-Spline interpolation. As in 2D, we use *Non-uniform Rational B-Splines* or NURBS

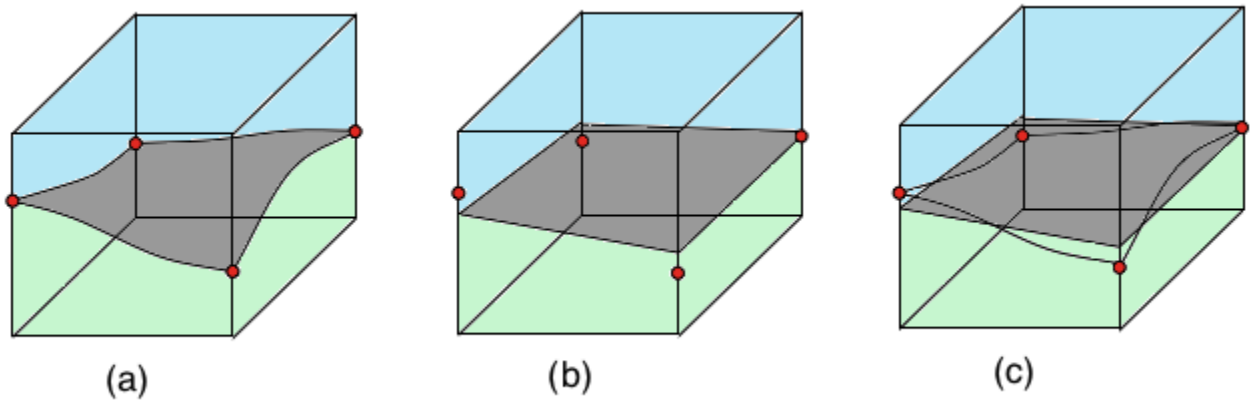


Figure 4.10: Planar least squares fit surface approximation in cube.

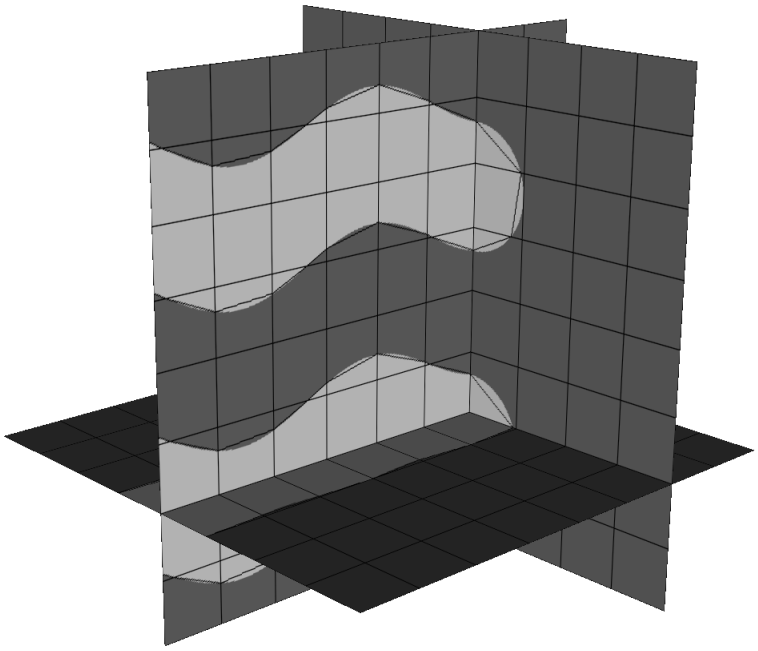


Figure 4.11: Planar least squares fit approximation for channels.

to create a mathematical representation of the geometry of the interface. The methodology employed requires that we first determine for each face of the element a NURBS approximating the interface curve on that face, and then use Coons' algorithm [20] to construct a Coons patch or Coons surface within the element.

Given four curves $C_1(u), C_2(u), D_1(v), D_2(v)$ that meet at the corners, two ruled surfaces can be obtained by linear interpolation between C_1 and C_2, and D_1 and D_2, respectively.

The four curves are defined using the NURBS basis functions

$$R_i^p(u) = \frac{w_i N_{i,p}(u)}{\sum_{j=0}^{n} w_j N_{j,p}(u)}$$

as

$$C_i(u) = \sum_{i=0}^{k} P_i R_i^p(u) \quad i = 1, 2,$$

$$D_i(v) = \sum_{i=0}^{k} P_i R_i^p(v) \quad i = 1, 2,$$

where $N_{i,p}(u)$ is defined recursively as in Chapter 2, Section 2.7.4, P_{ij} are control points, and w_{ij} are weights associated with these points.

The ruled surfaces obtained through linear interpolation are

$$S_1(u, v) = (1 - v)C_1(u) + vC_2(u),$$

and

$$S_2(u, v) = (1 - u)D_1(v) + uD_2(v).$$

The bilinearly blended Coons patch [46] is

$$E(u, v) = S_1(u, v) + S_2(u, v) - T(u, v),$$

where $T(u, v)$ is the bilinear tensor product surface

$$T(u, v) = C_1(0)(1 - u)(1 - v) + C_1(1)u(1 - v) + C_2(0)(1 - u)v + C_2(1)uv.$$

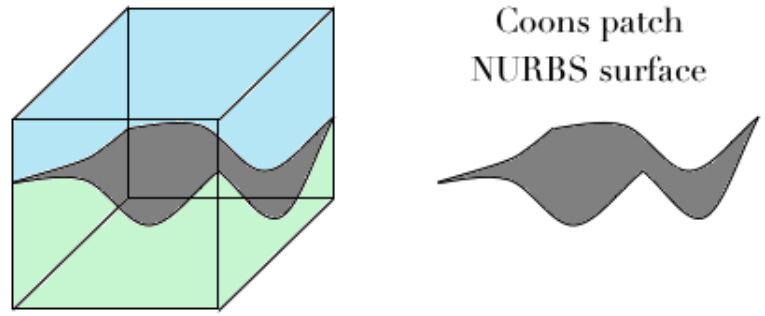


Figure 4.12: Coons surface

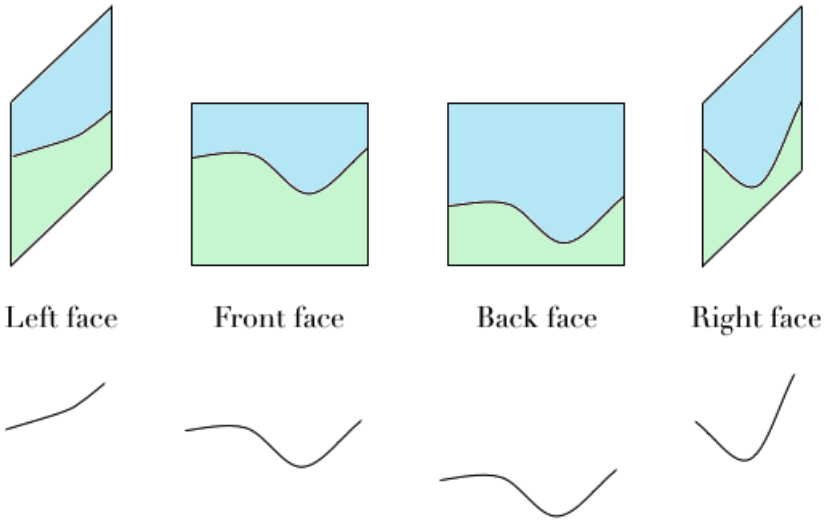


Figure 4.13: A decomposed brick element into its faces and corresponding interface intersections.

Figure 4.12 shows a hexahedral element with an interface that splits it in two. In order to obtain the surface approximation, we first need to determine the Coons patch $E(u, v)$. Figure 4.13 shows each face of the hexahedral element intersected by the interface, and the corresponding curves for each of these. These curves are the C_1, C_2, D_1, D_2 curves presented above.

Better surface approximations can be achieved by using bicubically blended Coons patches, but this requires that additional points on the surface be determined. These interior points are part of a *point cloud*, and are generally preferred to be uniformly spaced throughout the surface. Based on the various ways that a surface can intersect a hexahedral element, the size of the point cloud can vary. For quadratic NURBS that are bilinearly blended into Coons

patches, the size varies from 16 points (for 3 and 4 intersection points with the element) to 28 points (5 and 6 intersection points with the element).

Figures 4.14, 4.15, and 4.16 show the meshing for channels, tin battery, and microvascular channel, respectively, with NURBS interface approximation. For the channels (Figure 4.14), we show both a coarse mesh, and a finer one. The finer mesh has all constraints applied. For the tin battery (Figure 4.15), we show the mesh with only h-refinement applied. Some of the inclusions were not detected, either because the minimum element size was reached, and thus the element was declared homogeneous, or the inclusion is not of the significant size declared, which also made the element appear homogeneous. The microvascular channel (Figure 4.16) is also shown with only h-refinement. Since the channel is not aligned with any of the axes, we could not take orthogonal slices that show its full representation.

4.5.3 Comparison of Planar and NURBS Approximations

Tables 4.2, 4.3 and 4.4 can also be used to examine the effect of the interface approximation on the overall mesh size. Depending on the geometry of the structure, the difference between the two choices of planar approximations or B-Spline interpolation can be small or large.

In the case of the Fibers dataset (Table 4.2), the difference is quite dramatic. The mesh size of the planar approximation after the steep-gradient constraint is already about the size of the mesh using NURBS with all constraints applied. The final size of the mesh with planar approximation is very large. The ratio between planar and NURBS meshes for the cumulative factor (CF) is roughly three after each constraint, except the last one, where the ratio is closer to 1.35.

For the tin battery (Table 4.3), the NURBS mesh barely outperforms the planar least fit: 10853 elements versus 10970 after all constraints. This is also reflected in the cumulative factors between the various constraints. At each step, they are roughly the same. This could be due to the image resolution and size of the inclusions with respect to the size of the image. This small difference between the two approximations was also observed in the 2D meshing of a slice of the battery. Table 2.8 showed that when using NURBS there were 136 elements created for that specific 2D slice, versus 151 when using linears or quadratics.

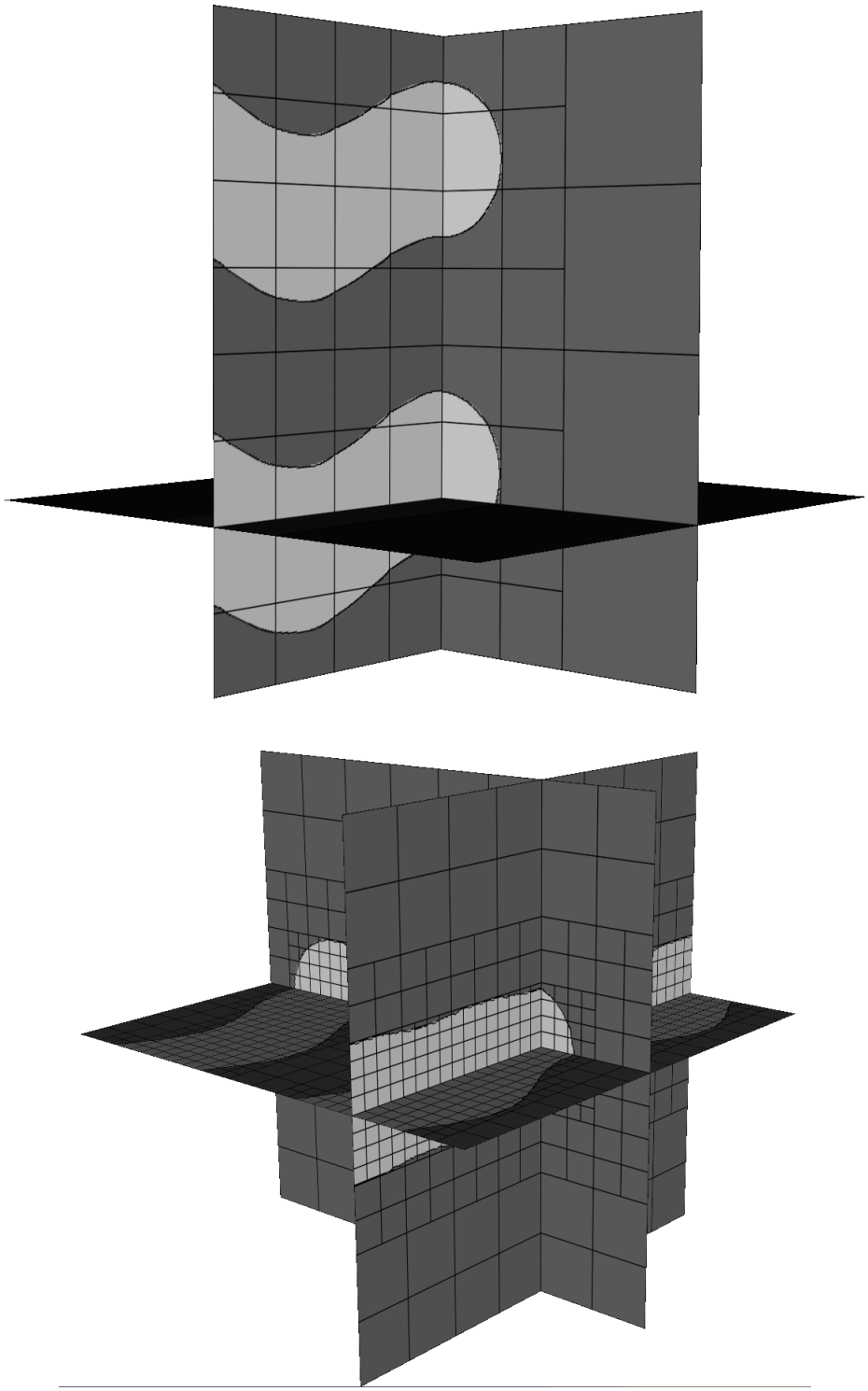


Figure 4.14: NURBS approximation for channels: coarse mesh above and finer one below. Finer mesh has all constraints applied.

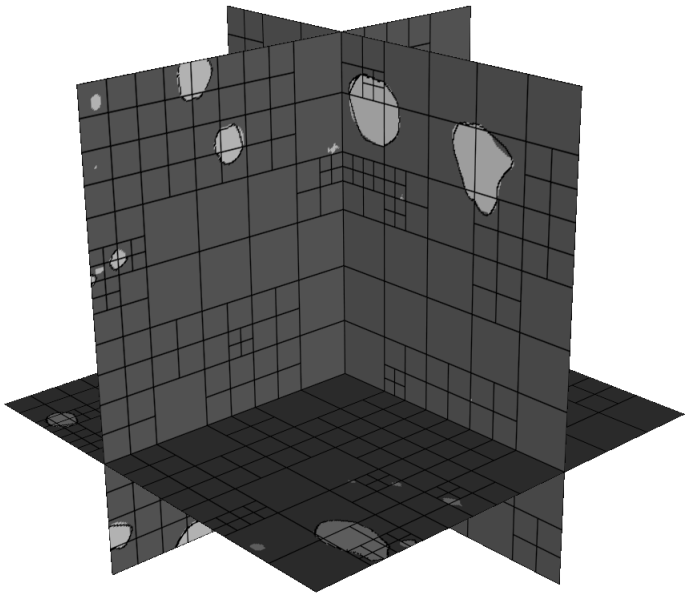


Figure 4.15: Mesh with NURBS approximation for tin battery.

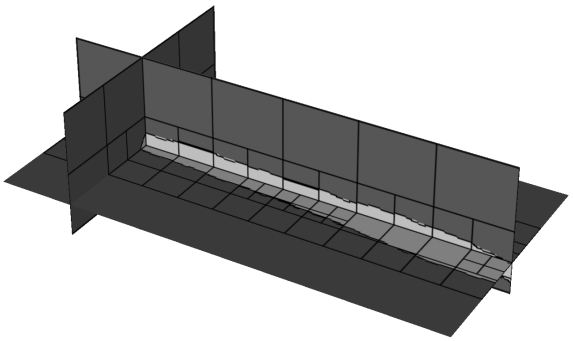


Figure 4.16: Mesh with NURBS approximation for microvascular channel.

Cubics created the same amount as NURBS, 136.

The mesh with NURBS for the microvascular channel (Table 4.4) starts slightly smaller after the first two constraints: about 161 fewer elements after h-refinement, and only nine fewer after the interface constraint. By the time the 1-irregular rule is applied, the mesh is of the same size as the mesh with planar approximations, and from then on until all the constraints are applied the two meshes remain identical.

Although for our datasets going from planar approximation to NURBS interpolation makes relatively little difference in the overall mesh size, we expect NURBS to be a better choice for 3D interface-based generalized finite element analysis. Having additional information about the curvature of the interface surface inside an element can only help the finite element analysis, especially for relatively complex interface morphologies. Both approaches, planar and NURBS can give an accurate solution, but for more complicated interfaces, NURBS provide equivalent accuracy at lower costs.

CHAPTER 5

Summary and Future Directions

5.1 Summary

In this thesis we have developed a meshing methodology for use in conjuction with IGFEM (the Interface based Generalized Finite Element Method) based on tomographic representation of heterogeneous materials. We extended IGFEM to use quadrilateral meshes in 2D. We developed the necessary meshing capability in both two and three dimensions.

The resulting finite element mesh is structured, non-conforming to the internal geometry of the inclusions, adaptive, and composed of quadrilaterals (2D) or hexahedrons (3D). In the context of tomographic data, the resulting mesh is pixel/voxel-aligned, and a subset of the image data. The meshing methodology is governed by requirements imposed by the classical Finite Element Method, as well as by IGFEM-specific constraints. These constraints dictate regions of mesh adaptivity, mesh size, how well the interface is approximated, and which homogeneous elements need to be subdivided. The algorithm is based on a divide-and-conquer approach and utilizes quadtree/octree generators.

The first constraint applied to the mesh is hq-refinement. Starting with the image dataset as one large element, recursive subdivision is triggered in the element until the maximum allowed element size is reached (h-refinement), and concomitantly determines whether the element is homogeneous or heterogeneous. The heterogeneity test is done by checking whether the element corners are in the same material, and then, if necessary, randomly sampling the interior of the element for significantly large inclusions. For each heterogeneous element, we approximate the interface inside it with a polynomial of desired degree (q-refinement) in 2D, or with a planar least squares fit in 3D, or with B-Splines in either dimension. The second constraint applied is that of limiting the number of nodes created by the intersection of the

interface with the element to two and not allowing any surrounding elements of a heterogeneous element to be subdivided. The third constraint is balancing the quadtree/octree to limit the number of subdivisions neighboring elements can have. The fourth constraint limits the number of neighbors of homogeneous elements that are subdivided. The last constraint is that of resolving steep-gradients in the vicinity of interfaces, by increasing the number of homogeneous elements between inclusions. This last constraint invalidates some of the previous ones, requiring them to be reapplied.

The initial two constraints are imposed by IGFEM. The remaining ones are also relevant to classical finite element meshes. For the last constraint, the steep-gradient constraint, we proposed a new approach that refines the mesh prior to any error analysis. Our proposed algorithm employs normal vectors to the interface and searches along these vector direction for the number of homogeneous elements between interfaces. If that number does not satisfy a preset tolerance, subdivision is triggered in these elements until that number is reached.

In both 2D and 3D we examined the cost of each constraint. After the first constraint is applied and up to the steep-gradient constraint, the mesh can double or triple in size in 2D, or can increase two-fold to six-fold in 3D. After the steep-gradient constraint is applied, the mesh can increase as much as 25 times in 2D or 87 times in 3D. This last constraint can be the costliest of all, depending on the specific problem geometry.

In localizing and approximating the shape of the interface, we also explored non-uniform rational B-Splines (NURBS). In both dimensions NURBS lead to smaller meshes than when using Newton polynomials up to degree three (2D), or planar least squares fit (3D). We particularly prefer them in 3D as they give an interpolation, not an approximation to the interface surface. The cost in 3D is that of applying a 2D algorithm to each face of a heterogeneous element and then using a Coons patch to build the interior surface. In localizing the interface we used standard image segmentation techniques combined with searches for change in pixel/voxel color along rays shot through the element at certain locations dictated by IGFEM.

We used the meshing algorithm with various pixel/voxel defined datasets that were either artificially created or created by an X-ray tomograph. These datasets represent composite materials that have either fibers, microvascular channels, or battery electrodes inside them.

To deal with noise in the images, we created bins of pixels/voxels that categorized each type of material (inclusion or surrounding) in ranges of pixel/voxel colors. We also employed for some of the images a smoothing algorithm both to delineate among inclusion boundaries and to discard inclusions too small to be significant in the overall composite.

We tested our 2D meshing algorithm by extending IGFEM to work with quadrilateral meshes. These meshes had the geometry of the interface inferred directly from images representative of composite materials. When using polynomial approximation of the interface, with hq-refinement we created heterogeneous elements with curved edges. The approach we used in dealing with these elements was to transform them to isoparametric elements that can be more easily integrated with Gaussian quadrature. We allowed the polynomial degree to go up to three and we observed the trade-off between h and q for various images. For many of the image geometries that we tried, a quadratic polynomial suffices, with very few elements requiring a cubic polynomial approximation. However, in the meshes that have a significant number of elements with cubics, the number of elements with quadratics is almost nil. This means that the interface either has a complicated curvature that only cubics can accommodate, or has a curvature for which quadratics are already good enough. The need for higher order polynomials (higher than linear) declines as we go to larger and larger meshes, as more and more elements are small enough that the interface does not have a complicated curvature to approximate. In order to evaluate the correctness of our IGFEM solution, we compared our solution to one generated using the classical finite element method with a highly-refined conforming triangular mesh. We observed that meshes with up to cubic polynomials for interface approximation performed slightly better than those with up to quadratic polynomials or linear polynomials. We also observed superlinear (nearly quadratic) convergence when plotting the L_2-norm of the error versus the element spacing. A direct comparison of a conforming triangular FEM mesh with a nonconforming quadrilateral IGFEM mesh also revealed that for a given triangular mesh size the same L_2-norm error can be achieved with fewer elements by using a quadrilateral mesh.

Summary of Major Contributions

- Developed and implemented 2D and 3D algorithms to create structured, adaptive, and nonconforming quad/hex meshes.
- Determined interface presence, location, and representation based on pixel or voxel data.
- Incorporated mesh adaptivity driven by
  - how well interface is approximated: polynomials, B-Spline interpolation, or planar least squares fit,
  - IGFEM-imposed constraints.
- Adapted IGFEM to use quadrilateral mesh in 2D.
- Demonstrated meshing and IGFEM codes by solving 2D thermal test problem for various composite material geometries.

5.2 Future Work

We have identified several topics in this thesis that can be further researched and expanded upon, and we discuss them in this section.

When using NURBS to approximate the interface we currently use only quadratic NURBS. If these NURBS do not give a good approximation, h-refinement is triggered. Allowing other orders NURBS to be tried before refining the mesh (q-refinement), or allowing a combination of mesh refinement with higher order NURBS (hq-refinement) could result in a smaller mesh. Although q- and hq-refinements are not currently implemented in the mesh in either dimension when approximating the interface with NURBS, these are features we are interested in developing in the future.

The current detection and representation of interfaces is done locally and thus enforced only to be C^0 continuous across boundaries. Enforcing a C^1 approximation could create a

better representation. This would require building a global continuous function that would incorporate information about the interface from each element.

We also find great potential in parallelizing the meshing algorithm both in 2D and in 3D. In particular in 3D, the octree generator could have a significant speedup with high resolution images.

In implementing the IGFEM analysis we came across several alternatives for implementing hanging nodes in the discretization. We chose the simplest of these, averaging of the solution at edge nodes, and only slightly started to investigate the effect of hanging nodes and enrichment nodes. Our first results indicated that in the case of IGFEM, when enrichment nodes are present on surrounding elements with hanging nodes, the classical approaches are equivalent to hybrid (triangular-quadrilateral) meshes where hanging nodes are eliminated.

In dealing with noise we created bins of pixels and applied smoothing algorithms. However, these were done through trial and error, based on histograms of pixel/voxel colors. Automating this process would determine the bins boundaries in a pre-processing step, directly from the data, without any human input.

REFERENCES

[1] Amira. `http://www.amira.com`.

[2] Image segmentation techniques, Wikipedia. `http://en.wikipedia.org/wiki/Image_segmentation`.

[3] Insight registration and segmentation toolkit (ITK). `http://www.itk.org`.

[4] Simpleware. `http://www.simpleware.com`.

[5] Adjerid, S., Belguendouz, B., and Flaherty, J. E. (1999) A posteriori finite element error estimation for diffusion problems. *SIAM J. Sci. Comput.*, **21(2)**, 728–746.

[6] Ainsworth, M. and Senior, B. (1997) Aspects of an adaptive hp-finite element method: Adaptive strategy, conforming approximation and efficient solvers. *Computer Methods in Appl. Mech. and Engr.*, **150(1)**, 65–87.

[7] Aragon, A., Duarte, C. A., and Geubelle, P. H. (2010) Generalized finite element enrichment functions for discontinuous gradient fields. *Int. J. Numer. Meth. Engr.*, **82**, 242–268.

[8] Aragon, A. M., Soghrati, S., and Geubelle, P. (2012) In-plane deformation effect on the cohesive failure of heterogeneous adhesives: a multiscale analysis based on a 3D generalized finite element scheme. *J. Mech. Phys. Solids*, **61(7)**, 1600–1611.

[9] Babuska, I. and Melenk, J. (1997) The partition of unity method. *Int. J. Numer. Meth. Engr.*, **40**, 727–758.

[10] Babuska, I. and Osborn, J. E. (1983) Generalized finite element methods: Their performance and their relation to mixed methods. *SIAM. J. Numer. Anal.*, **20(3)**, 510–536.

[11] Bank, R. (1994) PLTMG: A software package for solving elliptic partial differential equations. user's guide 7.0. *Frontiers in Applied Mathematics*, **Philadelphia, SIAM**.

[12] Bank, R., Sherman, A., and Weiser, A. (1983) Some refinement algorithms and data structures for regular local mesh refinement. *Scientic Computing, Applications of Mathematics and Computing to the Physical Sciences*, **1**, 3–17.

[13] Belytschko, T. and Black, T. (1999) Elastic crack growth in finite elements with minimal remeshing. *Int. J. Numer. Meth. Engr.*, **45**, 601–620.

[14] Belytschko, T., Gracie, R., and Ventura, G. (2009) A review of extended/generalized finite element methods for material modeling. *Modelling Simul. Mater. Sci. Engr.*, **17**, 1–24.

[15] Bern, M., Eppstein, D., and Gilbert, J. R. (1994) Provably good mesh generation. *J. Comp. System Sciences*, **48**, 384–409.

[16] Bern, M. and Plassmann, P. (2000) Mesh generation. *Handbook of Computational Geometry. Elsevier Science*.

[17] Boyd, S. K. and Muller, R. (2006) Smooth surface meshing for automated finite element model generation from 3D image data. *J. of Biomechanics*, **39(7)**, 1287–1295.

[18] Buades, A., Coll, B., and Morel, J. M. (2010) Image denoising methods. a new nonlocal principle. *SIAM Review*, **52(1)**, 113 – 147.

[19] Carey, G. F. (1997) Computational grids: Generation, adaptation, and solution strategies. *Series in Computational and Physical Processes in Mechanics and Thermal Science*, **Taylor and Francis, New York**.

[20] Coons, S. (1967) Surfaces for computer aided design of space forms. Tech. Rep. Project MAC-TR 41, MIT.

[21] Daux, C., Moes, N., Dolbow, J., Sukumar, N., and Belytschko, T. (2000) Arbitrary branched and intersecting cracks with the extended finite element method. *Int. J. Numer. Meth. Engr.*, **48**, 1741–1760.

[22] Dolci, C., Salvini, D., Schrattner, M., and Weibel, R., Spatial partitioning and indexing. `www.gitta.info/SpatPartitio/en/text/SpatPartitio.pdf`.

[23] Duarte, C., Babuska, I., and Oden, J. (2000) Generalized finite element methods for three-dimensional structural mechanics problems. *Computers and Structures*, **77**, 215–232.

[24] Duarte, C. A. and Oden, J. (1996) An *h-p* adaptive method using clouds. *Computer Methods in Appl. Mech. Engr.*, **139**, 237–262.

[25] Esser-Kahn, A. P., Piyush, R. T., Dong, H., Patrick, J. F., Vlasko-Vlasov, V. K., Sottos, N. R., Moore, J. S., and White, S. R. (2011) Three-dimensional microvascular fiber-reinforced composites. *Advanced Materials*, **23**, 3654 – 3658.

[26] Farin, G. (1992) *Curves and surfaces for computer aided geometric design - a practical guide*. Academic Press.

[27] Fish, J. and Yuan, Z. (2005) Multiscale enrichment based on partition of unity. *Int. J. Numer. Meth. Engr.*, **62**, 1341–1359.

[28] Fries, T. P., Byfut, A., Alizada, A., Cheng, K. W., and Schroder, A. (2011) Hanging nodes and XFEM. *Int. J. Numer. Meth. Engr.*, **86**, 404–430.

[29] Gonzales, J., Sun, K., Huang, M., Lambros, J., Dillon, S., and Chasiotis, I. (2014) Three dimensional studies of particle failure in silicon based composite electrodes for lithium ion batteries. *Journal of Power Sources*, **269**, 334 – 343.

[30] Hassan, O., Morgan, K., Probert, E. J., and Peraire, J. (1988) Unstructured tetrahedral mesh generation for three-dimensional viscous flows. *Int. J. Numer. Meth. Engr.*, **39**, 549–567.

[31] Heath, M. T. (2002) *Scientific computing, an introductory survey*. The McGraw-Hill Companies, 2nd edn.

[32] Hoschek, J. and Lasser, D. (1993) *Fundamentals of Computer Aided Geometric Design*. AK Peters.

[33] Kongrauz, Y. and Belytschko, T. (1998) EFG approximation with discontinous derivatives. *Int. J. Numer. Meth. Engr.*, **41**, 1215–1233.

[34] Lancaster, P. and Salkauskas, K. (1986) *Curve and Surface Fitting: An Introduction*. Academic Press.

[35] Lewis, A. C. and Geltmacher, A. B. (2006) Image-based modeling of response of experimental 3D microstructures to mechanical loading. *Scripta Materialia*, **55**, 81–85.

[36] Lorensen, W. and Cline, H. (1987) Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics*, **21**, 163–169.

[37] Madi, K., Forest, S., Boussuge, M., Gailliegue, S., Lataste, E., Buffiere, J. Y., Bernard, D., and Jeulin, D. (2007) Finite element simulations of the deformation of fused-cast refractories based on x-ray computed tomography. *Computational Materials Science*, **39**, 224–229.

[38] Meyers, D., Skinner, S., and Sloan, K. (1992) Surfaces from contours. *ACM Trans. Graph.*, **11**, 228–258.

[39] Moes, N., Cloired, M., Cartraud, P., and Remacle, J. F. (2003) A computational approach to handle complex microstructure geometries. *Comput. Meth. Appl. Mech. Engr.*, **192**, 3163–3177.

[40] Moes, N., Dolbow, J., and Belytschko, T. (1999) A finite element method for crack growth without remeshing. *Int. J. Numer. Meth. Engr.*, **46**, 131–150.

[41] Moore, P. (2000) An adaptive h-refinement finite element method for parabolic differential systems in three space dimensions. *SIAM J. Sci. Comput*, **21**, 1567–1586.

[42] Morton, G. M. (1966) A computer oriented geodetic data base; and a new technique in file sequencing. *Technical Report, Ottawa, Canada: IBM Ltd.*, **01**.

[43] Niekamp, R. and Stein, E. (2002) An object-oriented approach for parallel two- and three- dimensional adaptive finite element computations. *Computers and Structures*, **80(3)**, 317–328.

[44] Patrick, J., Hart, K., Krull, B., Diesendruck, C., Moore, J., White, S., and Sottos, N. (2014) Continuous self-healing life cycle in vascularized structural composites. *Advanced Materials*, **26(25)**.

[45] Peraire, J., Peiro, J., Formaggia, L., Morgan, K., and Zieniewicz, O. (1988) Finite element Euler computations in three dimensions. *Int. J. Numer. Meth. Engr.*, **26**, 2135 – 2159.

[46] Piegl, L. (1995) *The NURBS book*. Springer, Berlin.

[47] Rannacher, R. (2002) Adaptive finite element methods for partial dierential equations. *ICM*, **3(1-3)**.

[48] Simone, A., Duarte, C., and der Giessen, E. V. (2006) A generalized finite element method for polycrystals with discontinuous grain boundaries. *Int. J. Numer. Meth. Engr.*, **67**, 1122–1145.

[49] Soghrati, S., Aragon, A. M., Duarte, C. A., and Geubelle, P. H. (2010) An interface-enriched generalized finite element method for problems with discontinuous gradient fields. *Int. J. Numer. Meth. Engr.*, **89**, 991–1008.

[50] Soghrati, S. and Geubelle, P. H. (2012) A 3D interface-enriched generalized finite element method for weakly discontinuous problems with complex internal geometries. *Comput. Methods Appl. Mech. Engr.*, **217-220**, 46–57.

[51] Soghrati, S., Thakre, P. R., White, S. R., Sottos, N. R., and Geubelle, P. H. (2011) Computational modeling and design of actively cooled microvascular materials. *J. Heat Mass Transf.*, **65**, 153–164.

[52] Strouboulis, T., Babuska, I., and Copps, K. (2008) The design and analysis of the generalized finite element method. *Computer Methods in Appl. Mech. and Engr.*, **75**, 503–532.

[53] Sukumar, N., Hang, Z., Prevost, J. H., and Suo, Z. (2004) Partition of unity enrichment for bimaterial interface cracks. *Int. J. Numer. Meth. Engr.*, **59**, 1075–1102.

[54] S.Y.Kim, Sottos, N., and White, S. (2014) Self-healing of transverse cracks in cross-ply composites using a microencapsulated solvent-based healing system. *American Society for Composites 29th Technical Conference, 16th US-Japan Conference on Composite Materials, ASTM-D30 Meeting*, **203**.

[55] Toohey, K. S., Sottos, N. R., Lewis, J. A., Moore, J. S., and White, S. R. (2007) Self-healing materials with microvascular networks. *Nature Materials*, **6**, 581–585.

[56] Wang, S. (1997) An a posteriori error estimate for finite element approximations of a singularly perturbed advection-diffusion problem. *J. of Computational and Applied Mathematics*, **87(2)**, 227–242.

[57] Yerry, M. and Shephard, M. (1983) A modified quadtree approach to finite element mesh generation. *Computer Graphics and Applications*, **IEEE 3(1)**, 39–46.

[58] Yoder, R. and Bloniarz, P. (2006) A practical algorithm for computing neighbors in quadtrees, octrees and hyperoctrees. *Proceedings of the 2006 International Conference on Modeling, Simulation, and Visualization Methods*, **WorldComp 2006, Las Vegas**, 74–81.

[59] Youssef, S., Maire, E., and Gaertner, R. (2005) Finite element modeling of the actual structure of cellular materials determined by x-ray tomography. *Acta Materialia*, **53**, 719–730.

[60] Zienkiewicz, O. (1977) *The Finite Element Method*. McGraw-Hill, New York, 3rd edn.